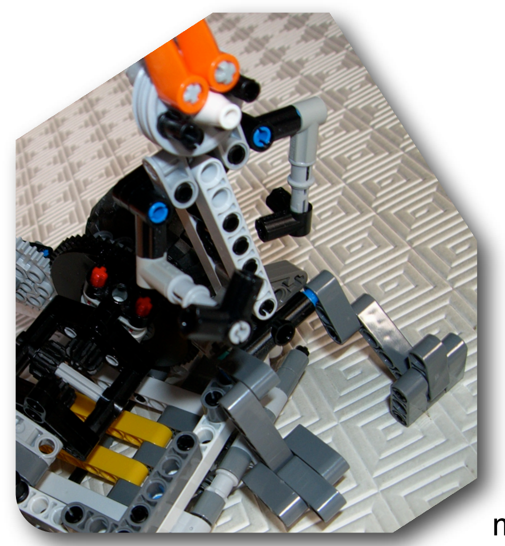


GUIDE de PROGRAMMATION NXT-G

Par Norbert Czaryski
http://web.mac.com/roboleo/Roboleo/Accueil.html

Adaptation et traduction partielles de l'ouvrage de James Floyd Kelly
«LEGO Mindstorms NXT-G, Programming Guide» édition Apress (http://www.thenxtstep.com)

Table des matières



Leçon n°1
INTRODUCTION A LA PROGRAMMATION

Bonjour à tous,
Ce personnage, c'est moi et je viens de terminer une partie de mon travail…
Alors je me repose un instant sur une ébauche de modèle (on appelle cela un MOC).
Il y a beaucoup de personnes, heureuses propriétaires du kit # 8527, qui souhaitent apprendre à mieux programmer le LEGO® Mindstorms® NXT.
Je vais donc essayer de vous instruire sur le sujet.

Mais tout d'abord, quelques petites règles de fonctionnement:
Moi, c'est Roboleo, mais j'accepte LEO.
Je ne suis pas un programmeur gourou, mais j'aime bien que mes programmes collent parfaitement avec mes MOC's. Certains sont élégants, d'autres affreux, mais tous fonctionnent correctement. Ils fonctionnent parce que j'ai consacré le temps nécessaire au bon usage des outils qui sont fournis dans le logiciel du NXT.

Première règle: familiarisez-vous avec les outils de programmation.
Et pour cela je vous propose des leçons qui s'adressent d'abord aux débutants.

•1 - Mais avant de commencer, précisons quelques évidences:
Vous avez acquis ou disposez d'un kit #8527.
Vous êtes sensés disposer d'un ordinateur et savoir vous en servir.
Naviguer sur le Net ne suffit pas.
Vous devez être en mesure de lancer un logiciel, et de procéder à toutes opérations d'enregistrement, sauvegarde, etc…
Vous avez installé le logiciel Mindstorms NXT sur votre ordinateur
Votre NXT est équipé de son microprogramme et il est prêt à fonctionner.
(Si vous ne savez pas, faîtes un tour ici http://setechnic.com/Forum/viewtopic.php?t=1256)

•Pour tester les programmes que nous allons élaborer, il nous faudra un robot de travail. Nous l'appellerons BONG. Confectionné le plus simplement possible, il comportera le NXT monté sur 2 roues motrices actionnées chacune par un servomoteur, + une roue "folle". Il sera équipé du capteur à ultrasons. Un exemple (sans capteur) avec vidéo et notice de montage est accessible ICI:
http://www.nxtprograms.com/five_minute_bot/index.html
Vous allez donc vous fabriquer un BONG.
LEO va ensuite communiquer de temps en temps avec BONG et BONG réagira.
J'espère qu'il comprendra les ordres qui lui seront adressés.

La première leçon commence ici:

Quelques définitions

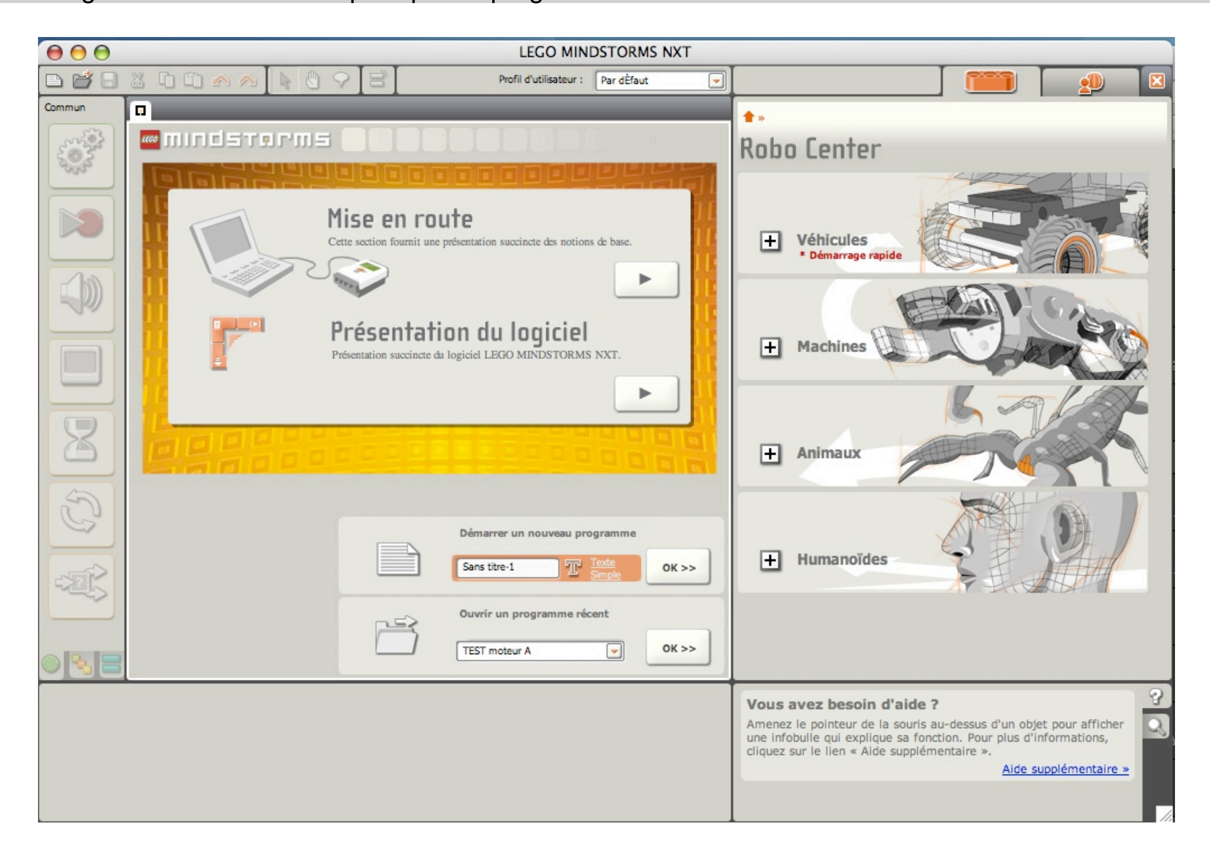
Un **ROBOT**: c'est un appareil construit pour accomplir d'une manière indépendante des actions et qui réagit avec son environnement.
Un **PROGRAMME**: Pour l'instant, BONG exécute ce qu'il fait le mieux, c'est-à-dire rien.
Pour le faire agir, LEO va lui donner des ordres, des instructions.

définition: un programme est une suite d'instructions. Programmer, c'est ce que fait LEO quand il crée un programme.
Cela paraît simple, vous verrez cependant qu'il n'en est rien…
Exemple:
LEO > BONG, avance
(BONG se déplace en avant)
LEO > BONG, stop
(BONG s'arrête)
LEO vient de donner à BONG deux programmes très simples à exécuter. Et oui, 2 programmes!
Le premier est "déplacer en avant", le 2ème "stop". Ils ne comportent chacun qu'une seule étape.

Est-il possible de réunir ces 2 étapes en un seul programme?
LEO > BONG, Avance et stop
(BONG ne réagit pas…)
Que se passe-t-il? Manque de précision. Plus exactement les choses se sont déroulées tellement vite que LEO n'a rien vu. Le moteur n'a même pas eu le temps de démarrer que déjà le STOP est intervenu. BONG a fait exactement ce que LEO lui a demandé.
Dans le premier cas LEO a attendu que BONG démarre, avant de lui demander de s"arrêter. Il a donc eu le temps de se déplacer.
Dans le 2 cas, LEO n'a pas précisé la longueur du trajet ou le temps du déplacement.
Essayons autrement.
LEO > BONG, avance pendant 5 secondes, puis stop.
(BONG se déplace en avant pendant 5 secondes, puis s'arrête).
OK, possible que BONG ne soit pas le problème. Cela signifie que BONG n'a aucun talent de devin et qu'il faut être très précis avec lui.

Voyons maintenant les choses différemment. Au lieu de lui dire comment agir, prenons une feuille de papier et écrivons:
BONG, avance de 90 cm; tourne à gauche de 90°; recule de 60 cm; fait un tour de 360° sur toi-même, et stop.
LEO donne cette feuille de papier à BONG qui la lit.
Devinez la suite?… BONG avance de 90 cm; recule de 60 cm; fait un tour complet sur lui-même et s'arrête.
Si votre Robot NXT est comme le mien, alors il est incapable d' entendre des ordres vocaux ou lire une feuille de papier.

Comment dans ces conditions lui transmettre les instructions?
Facile, vous allez utiliser le logiciel de programmation NXT, celui fourni dans le kit #8527 appelé NXT-G (G pour "graphique" signifiant que les suites d'instructions ne sont pas écrites comme il a été dit précédemment, mais représentées autrement).
Et voici ce que vous découvrirez après avoir lancé le programme:



NXT-G est l'outil de programmation que vous emploierez pour donner vie à vos robots. Il vous permet de créer vos programmes qui seront téléchargés (installés) sur votre robot. Ces programmes peuvent-être de simples instructions comme "avancer de 60 cm et stop" ou bien plus sophistiquées.
Les robots NXT peuvent combiner une grande variété de moteurs et de capteurs, mais, sans un bon programme, il n'accomplira pas ce que vous attendez de lui. Vous n'aurez entre vos mains qu' un objet dans la confusion.



Vous êtes tous passés, je l'espère, par la 'MISE en ROUTE' de NXT-G.
Vous avez donc vu l'espace de travail et toutes les autres parties qui sont les outils à votre disposition.
Mais, revenons à BONG.
Vous avez là aussi construit **votre** BONG, selon mes recommandations. Il est peut-être différent du mien, mais qu'importe, je dirais tant mieux. En effet, mêmes différents des BONG peuvent agir de la même façon, à condition que les ports (vous savez ce que c'est, n'est-ce pas?) soient raccordés de la manière suivante:
2ème règle:
Les capteurs doivent être connectés aux ports d'entrée (1,2,3 & 4) et les moteurs doivent être connectés aux ports de sortie (A, B & C).
Si cette règle n'est pas respectée, au mieux, il ne se passera rien ; au pire, je n'en sais pas plus.

Structure et programme

Je n'utiliserai pas ici de termes techniques, mais pour une fois, acceptez que je vous parle de structure de programmation.
Si les êtres humains sont capables par eux-mêmes d''accomplir certaines tâches non dites, rappelez-vous que BONG n'est pas aussi intelligent! Les instructions qu'il reçoit doivent lui être données d'une manière stricte et spécifique. De plus elles doivent être ordonnées.Cet ordonnancement est une façon de parler de structure de programmation.

Prenons un exemple:
BONG est immobile, en attente.
Quant à nous, nous ne sommes pas encore prêts à télécharger un programme NXT-G, mais nous pouvons pré-plannifier les tâches d'un futur programme.
Nous allons utiliser ce que certains programmeurs appellent un *pseudo-code*.

Pseudo-code? C'est tout simplement un 'faux programme'.
Ce pseudo-code ne sera pas écrit à l'aide du NXT-G, mais pour le faire nous supposerons que BONG a des oreilles et qu'il est capable d'enregistrer les ordres donnés.
Essayons d'écrire des pseudo-codes suivant une liste numérotée.
1. BONG, avance jusqu'à ce que le capteur tactile soit appuyé et relâché; puis stop.
2. Ok BONG, tourne maintenant à gauche de 90°.
3. Bon travail, BONG. Maintenant recule jusqu'à ce que le capteur photosensible détecte un objet noir; puis stop.
4. Maintenant BONG, danse un peu.

Ce que je veux dire par là, c'est qu'avant de programmer avec NXT-G, vous devez avoir une idée précise de ce que BONG doit faire. Et la meilleure façon de l'exprimer, c'est d'écrire en simple langage les instructions destinées à BONG.

En résumé, quand vous écrivez un pseudo-code, vous faîtes 2 choses:

- •Vous bénéficiez d'une meilleure compréhension du travail que BONG doit accomplir.
- •Vous créez un enclenchement logique de tâches (structure) que BONG doit suivre.

Une dernière remarque: pensez à ce que chaque instruction destinée à BONG soit la plus simple possible.
(LEO > Simple is beautiful…)

Pour appuyer ce que je viens de dire, dites-moi quel est l'exemple qui vous paraît le mieux convenir?

Exemple 1: BONG, avance de 30 cm; tourne à gauche de 90°, puis recule; ensuite cherche un objet noir à l'aide du capteur à ultrasons, parce que je veux un stop si tu trouves cet objet; alors tourne à droite de 90°, et recule de 50 cm, OK?
Exemple 2:

- •BONG, avance de 30 cm et stop
- •Maintenant tourne à droite de 90°
- •Recule, et active le capteur à ultrasons.
- •Stop, quand tu trouves un objet noir
- •Tourne à droite de 90° puis stop
- •Maintenant recule de 50 cm et stop

Vous avez trouvé?
Bien sûr, c'est le 2…Il faut dire que même un humain serait perplexe avec le 1!
Plus les instructions seront simples et courtes; et plus il vous sera facile de les traduire en NXT-G

Les Blocs dans NXT-G;

Reprenons le pseudo-code 3 ci-dessus:

" Maintenant recule jusqu'à ce que le capteur photosensible détecte un objet noir; puis stop."

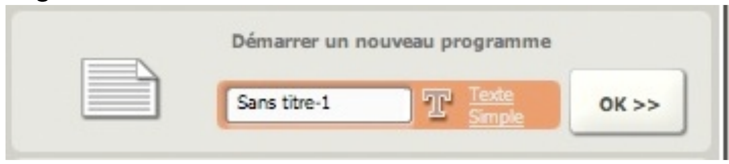
Si je programme en NXT-G et si les outils me sont familiers, je constate que certains collent parfaitement avec mon pseudo-code.
Quand je demande à BONG de reculer, il utilisera ses moteurs, non?
Bien, j'utiliserai alors un outil qui s'appelle un bloc 'DEPLACER'.
Ce bloc DEPLACER va me permettre de programmer la rotation des moteurs (et des roues) en sens inverse, et BONG reculera.
Je veux que BONG recule seulement jusqu'à ce le capteur photosensible détecte la couleur noir? J'emploierai l'outil bloc Capteur PHOTOSENSIBLE qui réagit à la lumière. Ce bloc sera programmé pour détecter la couleur noir.
Enfin, je veux que BONG stoppe quand le bloc capteur détecte la couleur noir. J'utilise pour cela à nouveau un bloc DEPLACER qui dira aux moteurs d'arrêter la rotation.

Vous utiliserez ces blocs et bien d'autres pour programmer correctement BONG et les autres robots.
En conclusion, retenez de cela:

PRENEZ LE TEMPS D'ECRIRE LES PSEUDO-CODES EN TERMES COURTS et SIMPLES, LA PROGRAMMATION EN SERA FACILITEE.

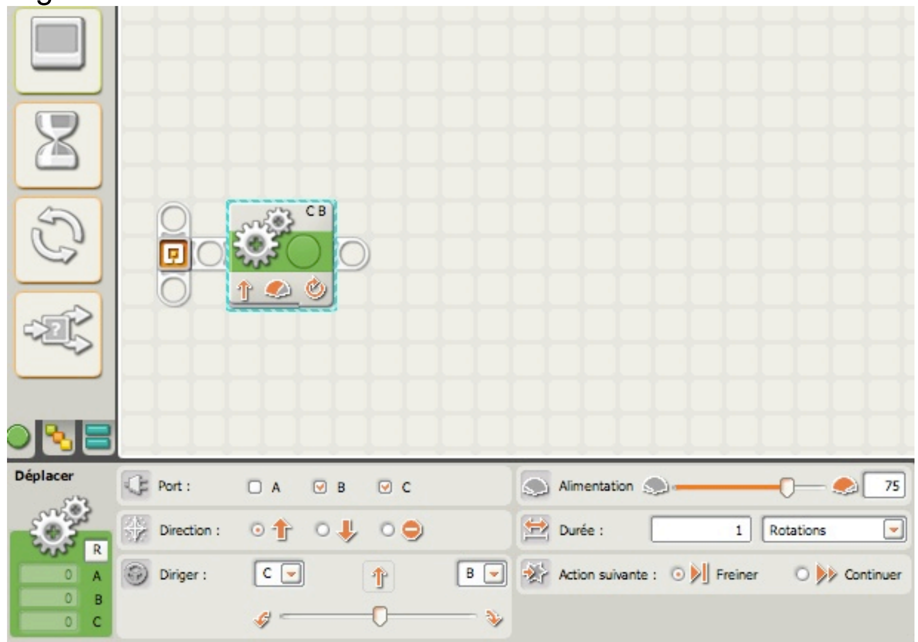
Nous allons aborder maintenant l'utilisation du BLOC DEPLACER, qui est le plus important dans la programmation d'un robot. Sans ce bloc, vous pourrez construire des modèles, mais ils seront très limités et ne feront pas grand chose. Ils seront immobiles et peu intéressants.
Tous les robots que vous concevrez et qui comportent un ou plusieurs moteurs utiliseront le BLOC DEPLACER.
Lancez le logiciel NXT-G, si ce n'est fait, et dans la page d'accueil Selectionnez 'Démarrer un nouveau programme'

Fig.1



Vous ouvrez à présent une nouvelle fenêtre dans laquelle nous allons travailler.
Dans cet espace, faîte disparaître la fenêtre 'Robo Center' pour avoir plus de place, en appuyant sur la petite croix en haut et à droite.
Dans la palette 'Commun' survolez la première icône 'Déplacer'.
Cliquez dessus et glissez vers le point de départ de l'espace de travail. L'icône vient se coller 'magnétiquement' à l'emplacement 'Démarrer'. Elle est encadrée d'un pointillé bleu qui indique qu'elle est sélectionnée.

Fig.2



Observez maintenant la fenêtre du bas. Elle est très importante: c'est le **panneau de configuration** du 'bloc Déplacer.
Chaque fois que vous sélectionnerez un bloc dans la zone de travail, son panneau de configuration sera automatiquement affiché. Bien entendu il est différent pour chaque bloc.

C'est dans ce panneau que le paramétrage du 'bloc Déplacer' se fera. C'est ici que vous choisirez les caractéristiques des moteurs que vous voulez contrôler. C'est une sorte de tableau de bord.
Les indications qui apparaissent sont celles de défaut. Ce bloc est déjà opérationnel. Si les paramètres ne vous conviennent pas, vous pouvez les modifier.

Examinons le détail. Vous remarquerez que chaque zone est caractérisée par une minuscule icône qu'on retrouve sur le bloc.
La zone **Port** indique les moteurs que vous voulez contrôler. Ici B et C.
On peut les modifier en cochant/décochant les cases correspondantes. Si on conserve B et C, il faut que les connexions entre les moteurs et le NXT soient respectées.
Dans ce cas les moteurs seront synchronisés et tourneront en marche avant ou en marche arrière à la même puissance. Si vous sélectionnez trois moteurs, les moteurs B et C seront synchronisés.
La zone **Direction** indique si les moteurs doivent tourner en marche avant ou en marche arrière, ou s'arrêter.
Si vous utilisez deux moteurs pour conduire un véhicule (un de chaque côté), la glissière **Diriger** s'affiche ; les lettres des ports sélectionnés sont indiquées de chaque côté.
Déplacez la glissière si vous voulez définir un trajet incurvé pour votre robot. Si la glissière est tirée entièrement d'un côté, votre véhicule tournera sur place.
La zone **Alimentation** dispose d'une glissière de saisie permettant de définir le niveau de puissance (0-100%), c'est à dire la vitesse.
La zone **Durée** dispose d'un menu déroulant pour indiquer si les moteurs doivent tourner pendant une durée illimitée ou pendant un nombre précis de rotations (par défaut), de degrés ou de secondes. Les options Temps, Rotation et Degrés permettent de déterminer la distance que votre robot parcourra.
La zone **Action suivante** Indique si les moteurs doivent freiner ou continuer en roue libre après avoir terminé leur action. Choisissez le freinage si vous voulez qu'ils s'arrêtent à un endroit précis. En outre, le freinage permet d'éviter que votre robot ne recule lentement s'il se trouve sur une pente.
Avec tous ces paramètres, vous pouvez programmer vos moteurs comme vous l'entendez.

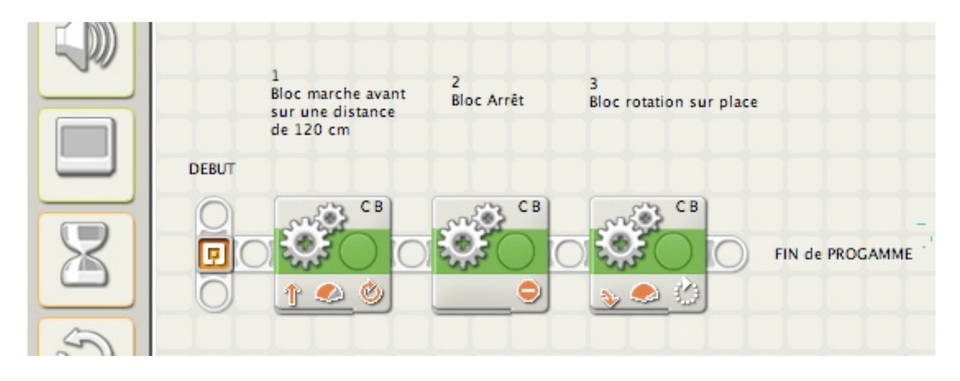
Observez à nouveau l'icône 'bloc Déplacer'; vous constaterez la présence des minuscules icônes qui rappellent le contenu du panneau de configuration. Changez les paramètres; ils seront modifiés sur le bloc.

Exercice:
reprenons l'exemple n°2 en modifiant quelques paramètres
* BONG, avance de 120 cm et stoppe
* Maintenant tourne à droite de 90°

Petit calculs préliminaires:
D = diamètre extérieur de la roue = 5,6 cm env.
Un tour de roue = 3,14 * 5,6 = 17,6 cm env.
Nbre de rotations = 120 / 17,6 = 6,8 env.

Le résultat se présente ainsi:

Fig.3



avec ses 3 panneaux de configuration:

Bloc 1



Bloc 2



Bloc 3



Vous allez maintenant sauvegarder ce programme.
Choisissez dans la barre de menus: >"Enregistrer le Programme" , et un fenêtre vous demande de lui donner un nom (BONG-1) ainsi que l'endroit du stockage puis validez.
Ou avez-vous enregistré ce programme? Evidemment sur votre ordinateur!

Il faut maintenant le télécharger sur BONG.
Allumez le NXT et connectez BONG à votre ordinateur (par câble USB ou Bluetooth), puis, téléchargez le programme (BONG-1).
Ce travail accompli, sélectionnez sur le petit écran du NXT "Software files", appuyez sur le bouton rouge et recherchez "BONG-1". Appuyez une 2ème fois sur le bouton rouge pour exécuter le programme.

On peut déjà en tirer quelques enseignements:
Il a fallut 3 blocs pour traduire le pseudo-code.
Les blocs s'exécutent l'un après l'autre dans l'ordre.
Le programme s'arrête quand il n'y a plus de bloc.

3ème règle:

Un bloc ne peut accomplir qu'une seule tâche à la fois. Il faut autant de blocs que de tâches à accomplir.

Nous allons maintenant détailler le contenu du bloc.

Les plots de données.

La plupart des blocs comportent un ensemble d'outils de connexion qu'on appelle PLOTS de DONNEES.
Ils sont généralement invisibles et il faut les activer pour les rendre visibles.

Fig.4

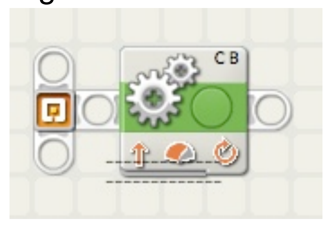
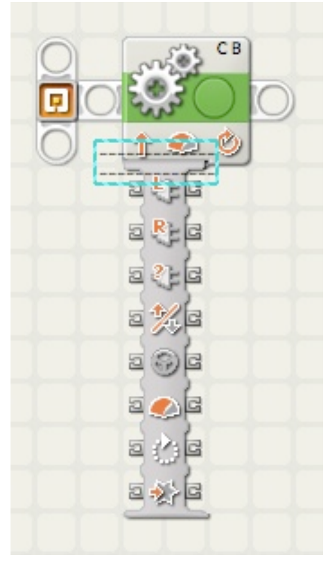
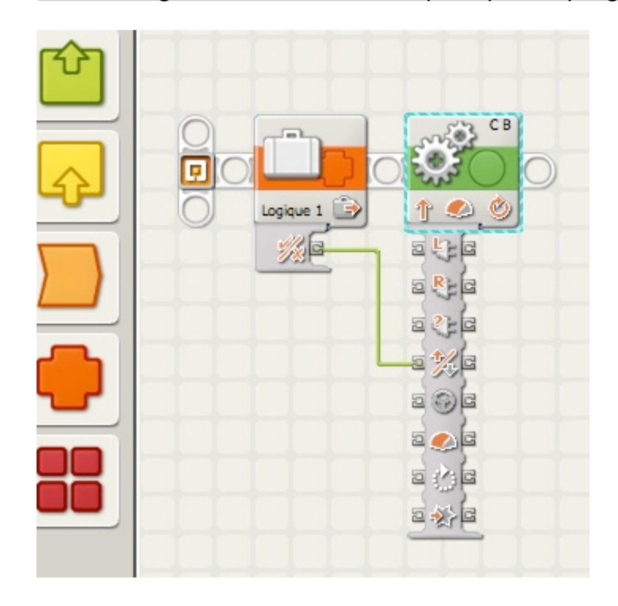


Fig.5
Il suffit de cliquer sur cette partie de l'icône pour dérouler la succession de plots:



Pour faire une comparaison, un peut dire qu'une "armoire électrique" vient de s'ouvrir.
Cette "armoire" comporte de part et d'autre de la zone centrale des "bornes" sur lesquelles vont se raccorder des "fils" comme des fils électriques.
La seule différence avec une installation électrique, c'est que ces bornes sont virtuelles et ne constituent que des points d'attache des fils (eux aussi virtuels) qui transportent des données. Mais chacun de ces points diffère de son voisin par la nature de l'information qui transite. On les reconnaît grâce à un repère graphique différent.
On remarque qu'il y a une "entrée" à gauche et une "sortie" à droite.
En voici un exemple:

Fig.6



Deux ou plusieurs blocs (pas forcément côte à côte) peuvent s'envoyer des informations si entre leurs bornes - leurs PLOTS - un fil les relie. Ces fils sont faciles à tracer; il suffit de pointer sur un plot, cliquer et le relier au plot souhaité.
Pour simplifier, on peut dire que les fils reliés à ces plots constituent un "câblage" particulier entre deux ou plusieurs blocs.
La manière de tracer ces fils et leur utilisation sera développée dans une autre leçon.
Pour l'instant ils sont inemployés, car sans objet.
Pour fermer cette "armoire", il suffit de cliquer au même endroit.

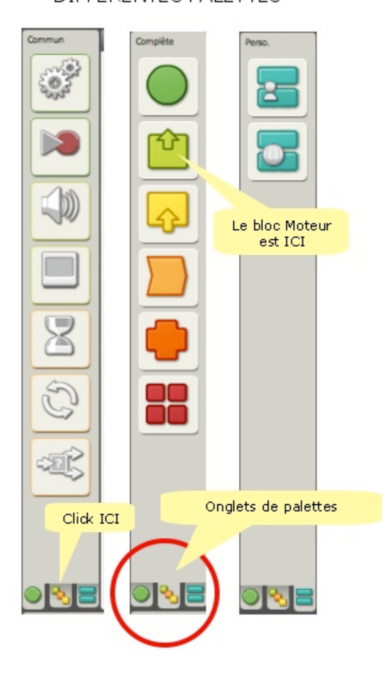
Vous avez appris à contrôler par programme, le fonctionnement des moteurs grâce au bloc '**Déplacer**'. Il en existe un autre appelé '**Moteur**'; il sera vu à l'occasion d' exercices.
Essayez d'autres combinaisons de blocs en modifiant les paramètres et testez à nouveau.

Les Palettes.

Nous allons maintenant compléter notre tour d'horizon avec le bloc '**Moteur**'.
Mais d'abord parlons de 'Palettes':
Il en existe trois offrant des collections de blocs complémentaires. Elles sont à disposition dans l'espace de travail.
Pour récupérer le bloc 'Moteur', vous allez sélectionner l'onglet central qui affiche la palette correspondante.
Cette palette est dite 'Complète' parce qu'elle dispose plus de blocs que la palette 'Commun' ; donc adaptée à la confection de programmes plus élaborés.

Fig.7

DIFFERENTES PALETTES



Bloc Moteur.

Nous allons comparer ce bloc 'Moteur' avec le bloc 'Déplacer'.



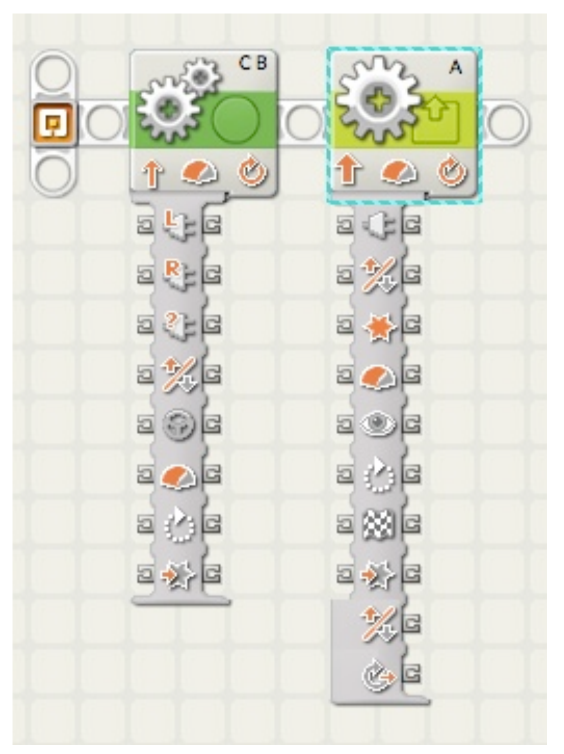
Observez attentivement les panneaux de configuration. A priori ils semblent identiques.

Le bloc « ***Moteur*** » est conçu principalement pour faire tourner et arrêter **UN** moteur. Vous pouvez préciser la durée du mouvement (nombre de rotations ou nombre de secondes pendant lequel il doit tourner), mais votre robot peut dépasser sa cible.
Ce dépassement peut survenir car le bloc « *Moteur* » n'est pas programmé pour ralentir alors qu'il atteint son objectif de durée, de telle sorte que son élan peut l'emmener au-delà de sa destination.
Examinons maintenant la rubrique *Action* dans le panneau de configuration.Le champ déroulant nous offre 3 choix : Constante, Accélération progressive, Ralentissement progressif.

La sélection *Constante*, signifie que le moteur tournera à vitesse constante, du début à la fin de son activation (telle que programmée dans le panneau de configuration). Cette disposition est intéressante quand le robot doit se déplacer à vitesse constante.
La sélection *Accélération progressive* déclenche la rotation du moteur d'une manière progressive jusqu'à ce qu'il atteigne la vitesse programmée ; c'est un peu comme une pédale d'accélérateur avec un limitateur de vitesse.
La sélection *Ralentissement progressif* entraîne le ralentissement progressif du moteur jusqu'à son arrêt. Cette disposition est soumise à une réserve. Elle peut-être utilisée seulement quand le bloc *Moteur* est précédé d'un autre bloc *Moteur,* réglé lui à vitesse *constante* et sur *Continuer* dans la rubrique *Action Suivante*.
Il faut une vitesse initiale pour permettre au bloc *Moteur* d'effectuer le ralentissement. Cela ressemble à un freinage de véhicule. Sans ce bloc *Moteur* le précédant, le *Ralentissement progressif* aura une vitesse nulle, et par conséquent sera incapable de ralentir.

Le bloc « ***Déplacer*** », peut faire tourner jusque 3 moteurs à la fois. Cela est visible sur les plots de données.
Il offre plus de souplesse. L'algorithme interne qui programme le NXT agit sur le tachymètre du moteur et ralentit la vitesse alors qu'il s'approche de son objectif de durée.
Cela est très utile quand votre véhicule doit suivre un tracé ou arriver à une distance très précise d'une cible.
Ce bloc ne contient aucune rubrique *Action* dans le panneau de configuration pour jouer sur les « comportements » du moteur. Cela est dû au fait que ce bloc contient déjà une certaine intelligence dans sa conception et qu'il intègre les fonctions de ralentissement et d'accélération.

Voyons à présent les plots de données:
Comme vous le constatez, Le bloc "Moteur" dispose de 2 plots particuliers, en "sortie" seulement. Cela signifie qu'il peut fournir des informations internes (sens de la marche, rotations en degrés), qu'on peut utiliser comme informations provenant d'un capteur (et oui, le moteur peut-être aussi un capteur).



Nous en reparlerons à l'occasion d'un exercice.

Nous avons à peu près fait le tour de la question en ce qui concerne les moteurs. La meilleure façon d'apprécier les différences, c'est de vous exercer en modifiant les paramètres et en testant les résultats sur les moteurs.
… Mais, avec les moteurs, ce n'est jamais fini. Il y a encore d'autres astuces, notamment des capteurs internes comme le chronomètre et la rotation. Nous en reparlerons à l'occasion d'une leçon sur les capteurs.



Rappel ici des règles

*** Première règle: familiarisez-vous avec les outils de programmation.**
*** 2ème règle: les capteurs doivent être connectés aux ports d'entrée (1,2,3 & 4) et les moteurs doivent être connectés aux ports de sortie (A, B & C).**
*** 3ème règle: Un bloc ne peut accomplir qu'une seule tâche à la fois. Il faut autant de blocs que de tâches à accomplir.**

La coutume étant de se présenter, les premiers mots seront pour donc pour la communauté du Mindstorms NXT.
LEO > BONG, aimerais-tu te présenter et dire qui tu es. J'aimerais que tu affiches "Salut de BONG!" sur ton petit écran.

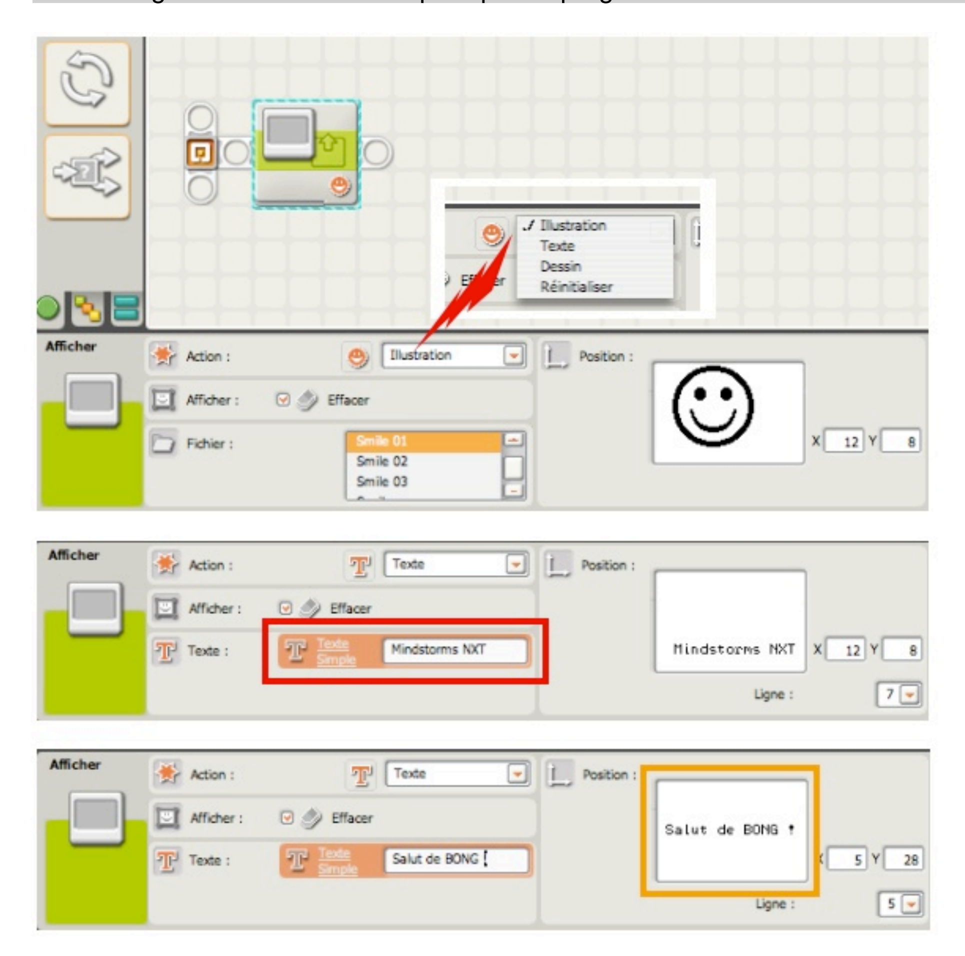
Afficher

Le programme ne peux pas être plus simple.
Pour cela nous allons utiliser le bloc '**AFFICHER**", à choisir dans la palette 'Commun'.
Voici à quoi ressemble ce bloc quand il s'affiche par défaut.
Dans la zone Action, vous avez un menu déroulant qui nous offre plusieurs options. Choisissons 'Texte'.

Le texte

Le panneau de configuration change d'aspect. Dans la zone 'Texte' encadrée de rouge, tapez à la place de 'Mindstorms NXT', 'Salut de BONG!'. Le texte est reproduit automatiquement dans la fenêtre encadrée d'orange qui est la fenêtre de prévisualisation..

Fig.1



Quelques précisions sur cette petite fenêtre: elle représente l'écran du NXT. Le texte que vous venez de taper se trouve sur la ligne 5 (sur les 8 que contient l'écran). Ses coordonnées cartésiennes indiquent que la ligne commence au point 5, à partir de la gauche et au point 28 en descendant du coin haut gauche (qui est l'origine des axes). L'écran est couvert d'un ensemble de points lumineux appelés Pixels constituant une sorte de grille.
Il a pour dimension 100 x 64 pixels. Ce n'est pas très grand, mais BONG n'est pas non plus un géant.
Si vous survolez cette zone à l'aide de votre souris, vous constaterez que le curseur prend la forme d'un main pointant du doigt. En cliquant, vous saisissez le texte qui peut être déplacé très facilement en n'importe quel endroit de la fenêtre. Par voie de conséquence, les indication chiffrée changent. C'est une manière aisée de positionner le texte, et surtout de visualiser ce que vous verrez sur l'écran.

Sauvegardez ce miniprogramme (nommé AFFICHER) sur votre ordinateur.
Allumez le NXT et connectez BONG (à votre ordinateur), puis, téléchargez le programme. Ce travail accompli, sélectionnez sur le petit écran du NXT "Software files", appuyez sur le bouton rouge et recherchez (AFFICHER). Appuyez une 2ème fois sur le bouton rouge pour exécuter le programme.

Avez-vous vu quelque chose?
Je ne pense pas. Le programme est supposé écrire 'Salut de BONG!' sur l'écran puis se termine.
L'exécution a été tellement rapide, que vous n'avez pas eu le temps de voir le texte s'afficher.

Quatrième règle: un programme NXT-G se termine une fois le dernier bloc exécuté.

La bonne nouvelle, c'est qu'il existe un moyen très facile pour le maintenir.
Le *pseudo-code* sera donc corrigé:
LEO > BONG, j'aimerais que tu affiches sur ton petit écran "Salut de BONG!" pendant 10 secondes.

Il existe de nombreuses manières pour maintenir un affichage; je vais vous en montrer une; vous découvrirez les autres par vous-mêmes en pratiquant.

Survolez la palette 'Commun', arrêtez-vous sur le bloc 'Attendre', puis dans la palette horizontale sélectionnez 'Temps'.

Fig.2



Ce bloc fait exactement ce qu'il dit: il attend.
Déposez-le à la suite du bloc "Afficher". Remarquez que le bloc "Afficher" a un T (pour texte) sur l'angle inférieur droit.

Fig.3



A présent, enregistrez et lancez le programme (AFFICHER MIEUX) -vous savez comment faire, n'est-ce pas?-
Cette fois ci, vous avez le temps de lire "Salut de BONG!" sur le petit écran!

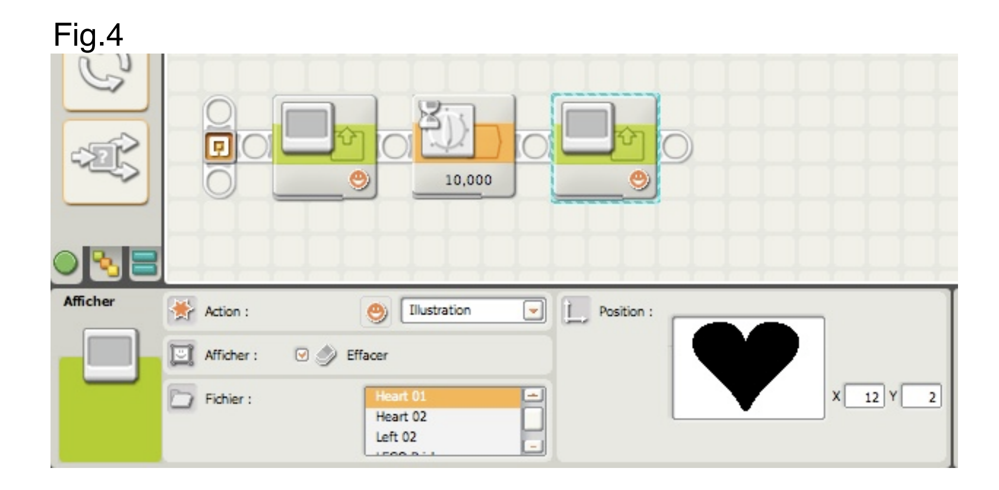
C'est un peu simpliste, mais vous avez appris à écrire un texte sur l'écran.
Remarque: l'écran peut recevoir jusqu'à 8 lignes de texte. Si l'on souhaite le remplir en totalité, il faudra autant de blocs que de lignes, soit 8 blocs.

L'image

LEO > BONG, montre maintenant à nos amis que tu as un coeur qui bat.

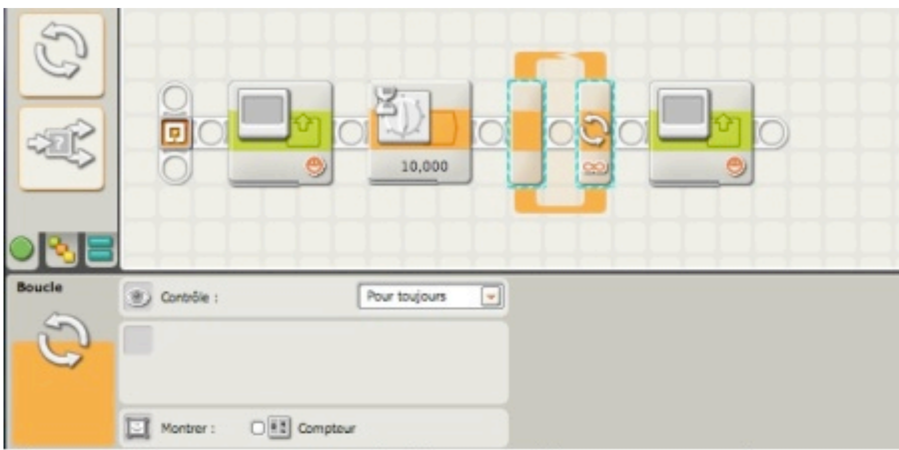
Nous allons procéder ainsi:
Ajoutez à la suite des blocs un nouveau bloc "Afficher".
Vous allez modifier le contenu du panneau de configuration et lui donner cet aspect.

Fig.4

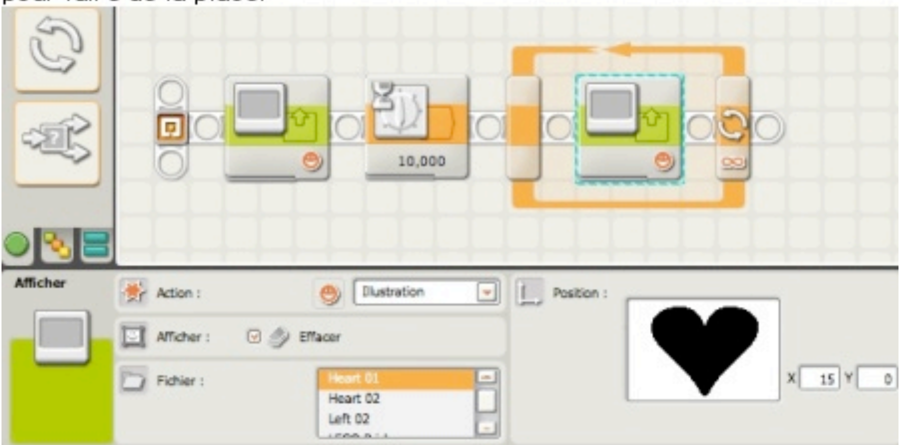


Rester sur "Illustration" dans la zone ACTION.
Dans la zone FICHIER, rechercher et sélectionnez dans le champ déroulant 'Heart 01'. Ce motif s'affiche dans la fenêtre de prévisualisation.
Rappelez vous, si j'exécute le programme ainsi, je n'aurai pas le temps de voir ce coeur; il faut donc lui adjoindre un délai. En prenant cette précaution, je ne verrai qu'un coeur statique. Or, je souhaite voir son coeur battre. Il faut donc faire de l'animation.
Comment?
En y ajoutant un autre coeur, plus petit, avec un autre délai identique. Mais, ce n'est pas encore fini…
Si j'exécute le programme à ce stade, je verrai le coeur gros, puis petit battre qu'une seule fois.
Insuffisant.
Il me faut donc répéter en permanence ce passage gros-petit, pour simuler le battement du coeur. Nous le ferons à l'aide d'un bloc "BOUCLE".
Vous allez maintenant composer la suite ainsi:

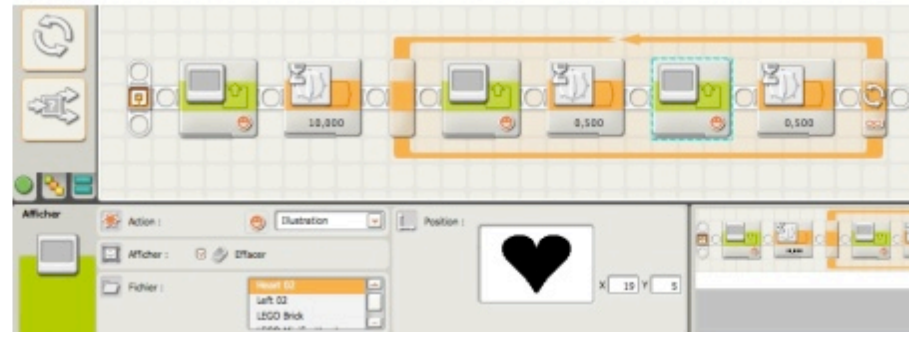
Fig.5



Glissez un bloc BOUCLE entre les blocs ; il s'écartent automatiquement pour faire de la place.

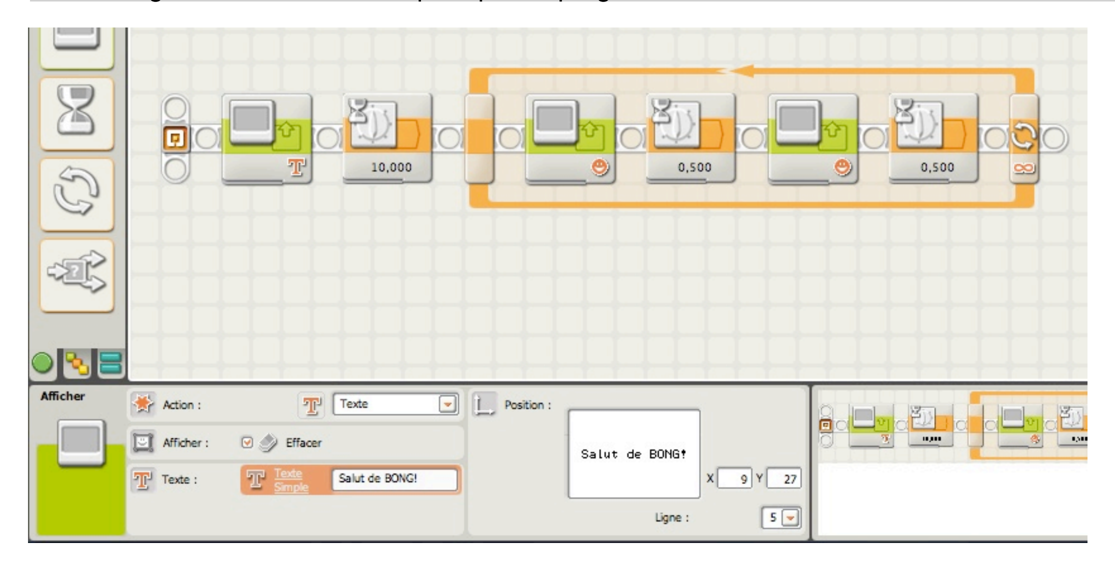


Glissez le bloc AFFICHER à l'intérieur de la boucle. Procéder ainsi pour les autres



Dans le choix du 2ème motif, selectionnez bien 'Heart 02", et réglez le temps à 0,5 seconde.
Il ne reste plus qu'à s'assurer que le premier bloc est bien un texte. Ici, ce n'est pas le cas; rectifions en conséquence.
Et voilà le résultat final

Fig.6

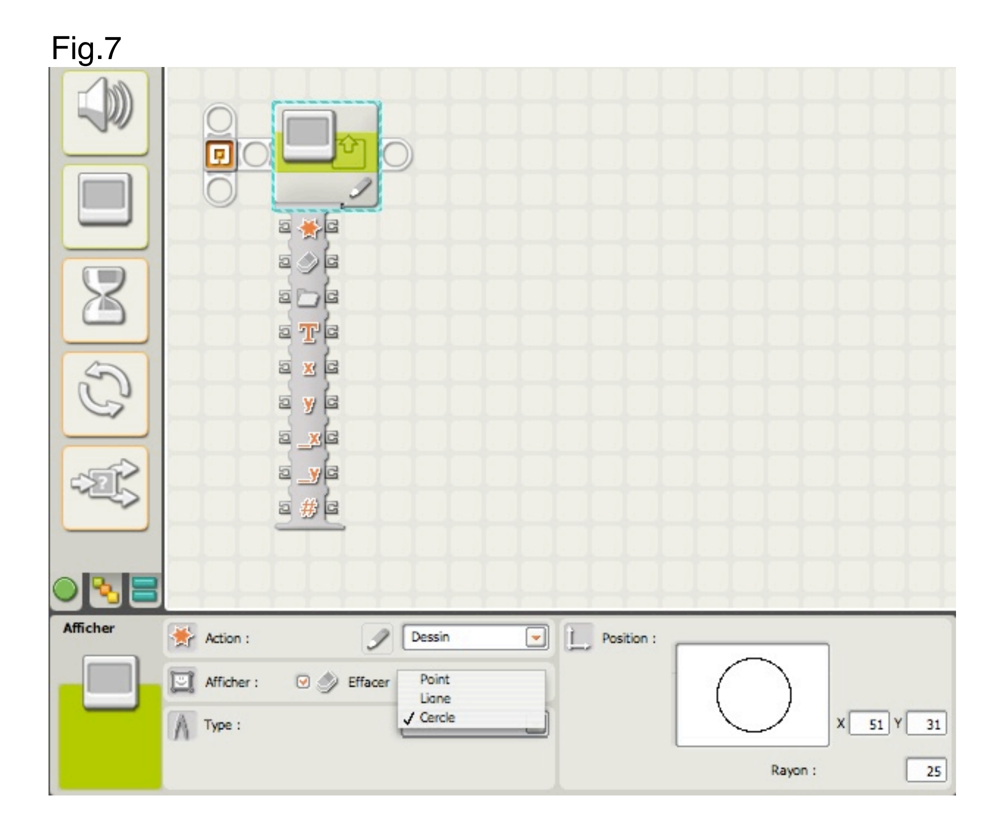


C'est terminé, enregistrez, exécutez ce programme (BONG coeur) et appréciez le résultat.

Nous avons traité dans la zone ACTION que 2 choix. Il en reste 2 autres.
Le choix **REINITIALISER** permet d'effacer tout l'écran. Cela est utile quand on ne souhaite pas retrouver dans la suite d'affichage des traces susceptibles de corrompre le résultat attendu. Il s'agirait de valeurs "oubliées" ou de caractères cachés par exemple. Ce choix permet donc de poursuivre en affichant un nouveau travail.

Enfin le choix **DESSIN** offre la possibilité de dessiner sur l'écran des figures géométriques. Ces dessins peuvent être statiques ou dynamiques, et s'ils découlent de formules mathématiques, ils peuvent être associés aux moteurs et capteurs. Dans ces cas, représenter par des tracés dynamiques les mouvements ou la présence d'objets, l'éloignement, etc…

Fig.7



Dans la zone Type, nous avons à notre disposition 3 outils; le cercle a été sélectionné.

On peut le déplacer dans la fenêtre de visualisation à l'aide de la souris ou en modifiant les coordonnées x,y.
Le rayon est aussi modifiable; donc adaptable à toute sorte de présentation. On peut aussi faire une animation en s'inspirant de l'exemple précédent: déplacement d'un cercle dans l'écran, grossissement progressif, etc…
Mais il y a une contrainte: il vous faut multiplier le nombre de blocs, ce qui alourdit le programme.
Tranquillisez-vous, il existe une méthode très élégante utilisant un bloc particulier "Variable" qui calcules les positions x et y. Nous en parlerons plus tard.
Les options **Point** et **Ligne** s'utilisent de la même manière.
Remarque: pour tracer un rectangle, il vous faut tracer 4 lignes, donc créer 4 blocs.

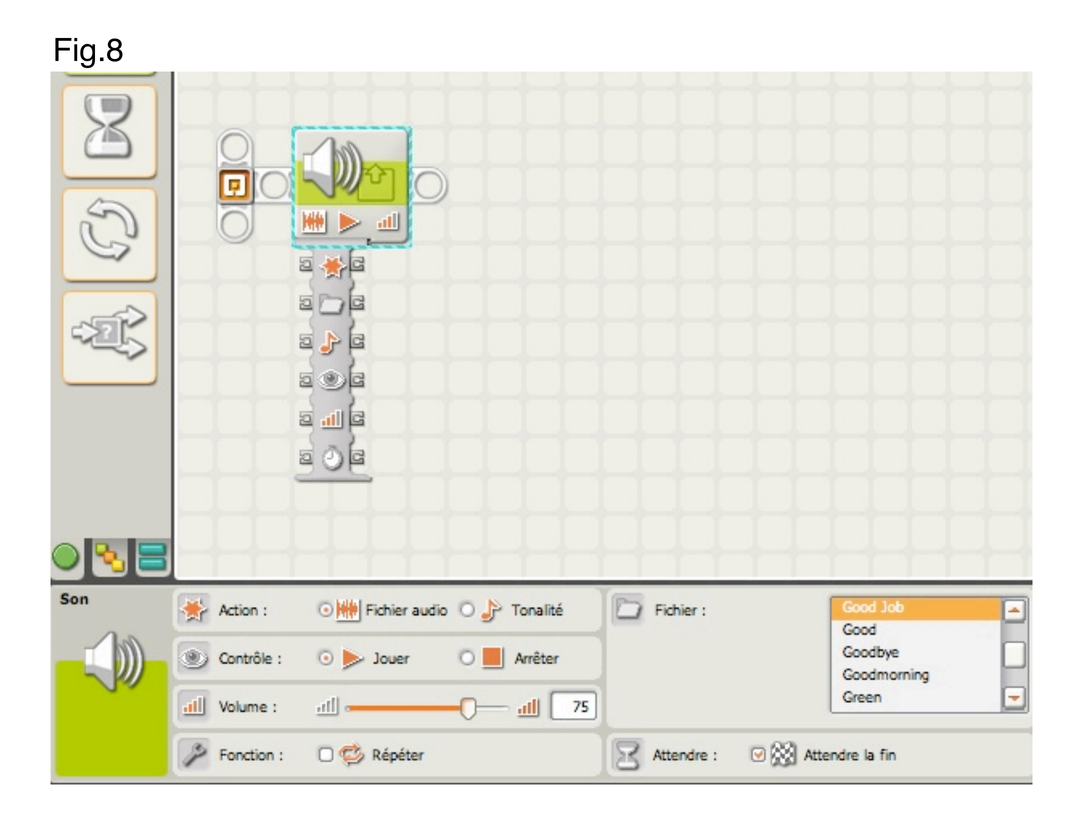
Juste pour mémoire, voici à quoi ressemblent les **Plots de Données**; vous pouvez constater que des plots sont prévus pour les coordonnées x,y. Comme le NXT est un calculateur, il est capable d'envoyer des valeurs de positionnement ou de dimension (à partir de variables), lesquelles seront reportées sur l'écran.

Le son

Vous avez testé le bloc "AFFICHER". Nous allons à présent donner un peu plus de personnalité à BONG, et pour cela nous allons introduire un nouveau venu: le bloc "SON". Vous pouvez faire parler BONG, mais vous pouvez aussi utiliser les sons pour évaluer la progression dans un parcours en employant des tonalités différentes. Ou tout simplement jalonner un programme par des tops sonores pour vérifier son bon déroulement, etc…

Le bloc "Son" se présente ainsi avec son panneau de configuration par défaut. Les plots de données sont également visibles pour la circonstance.

Fig.8



Dans la zone ACTION, 2 options sont disponibles. Le "fi*chier audio*" est sélectionné par défaut. Dans la zone "*Fichier*" un menu déroulant vous propose de choisir un ***fichier son*** préenregistré faisant partie d'une collection fournie avec le programme NXT-G. On y

trouve des bruitages, paroles (en anglais), sons jingle, et autres… Cette collection peut s'agrandir de vos propres enregistrements. Vous trouverez quelques explications en fin de cette rubrique.

Fig.9

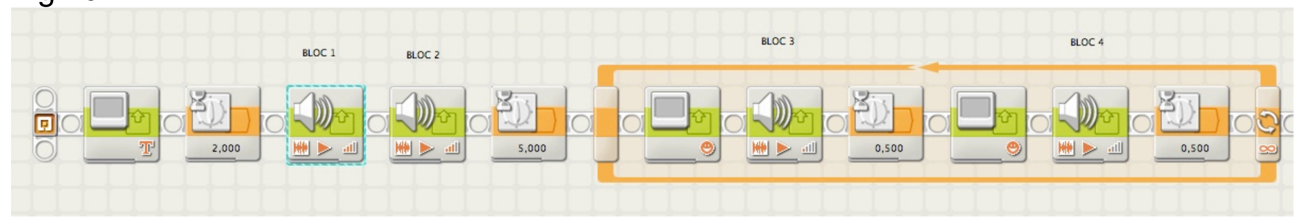


L'option "*Tonalité*" vous offre une série de réglages intéressants.
Dans la zone "*Remarque*" vous choisissez d'abord une **note** (le plus simple est de cliquer sur une des touches du clavier piano, qui émet aussitôt le son correspondant). Vous remarquerez que son nom s'affiche dans le champ de saisie (sous la forme de lettres A, B, C, D, E, F - pour La, Si, Do, Ré, Mi, Fa). Ensuite, précisez la durée. Vous remarquerez enfin que les zones du panneau de configuration situées à gauche sont identiques.
Si vous cochez "*Répéter*" dans la zone "*fonction*", la note sera maintenue jusqu'à la fin du programme, ou bien jusqu'à la lecture d'un autre "*bloc Son*" réglé sur "*Arrêter*" (sélectionné dans la zone "Contrôle").
Le "*Volume*" réglable, sera déterminé selon l'effet recherché. On notera cependant que plus le volume est élevé, plus la batterie d'alimentation est sollicitée.
Enfin, dans la zone "*Attendre*", si vous cochez "*attendre la fin*", le fichier son ou la note seront entendus en totalité, **avant** que le bloc suivant ne soit exécuté.

Pour conclure, reprenons le dernier programme (AFFICHER MIEUX) et procédons à quelques améliorations.

LEO > BONG, veux-tu dire "Hello Lego" et émettre le bruit des battements du coeur.
Glissez 4 blocs "Son" aux emplacements indiqués ci-dessous, en modifiant les blocs "Attendre".

Fig.10



Les blocs sont ainsi configurés

Fig.11



Vous savez maintenant ajouter des sons pour personnaliser encore plus votre robot, mais sachez aussi que les sons peuvent-être de précieux auxiliaires pour tester vos travaux. Placer un son à l'intérieur d'un programme complexe facilite vos recherches en cas de correction. De même, placer un son en tête d'un bloc moteur en donnant des tonalités différentes permet de vérifier le mouvement à droite, à gauche ou tout droit. Une fois les vérifications faîtes, il ne reste plus qu'à supprimer ces bloc sons.

Créer vos propres sons:

Ce paragraphe est destiné à ceux qui ont une bonne pratique de l'informatique sur PC.
Si vous possédez la version NXT 2.0 sautez ce paragraphe.

Soyez toujours prudents: pensez souvent à sauvegarder vos données.
Vous l'avez constaté, votre robot ne sait parler qu'anglais, et les bruits ou effets sonores sont limités.
Rien cependant ne vous empêche de personnaliser la collection de sons en y introduisant la vôtre.

Quelques remarques préliminaires:
votre NXT accepte seulement des fichiers dont les noms sont sous la forme ***monson.rso.***
Tous les ordinateurs ne savent pas créer forcément ce type de fichier, mais la plupart utilisent le format ***monson.wav.*** Ce format est celui qui servira à la conversion en fichier .rso.
La première démarche consiste donc à enregistrer vos nouveaux fichiers sons en .wav.

Si vous travaillez sous windows vous utiliserez sans doute **Sound Recorder.** Avec un Mac, si vous disposez de la version **QuickTime Pro**, vous pourrez convertir presque tous les fichiers sons en .wav.
Une grande variété de bruitages sont accessibles sur le Net, beaucoup sont gratuits; vous pourrez donc en profiter.

Pour créer vos bruitages vous utiliserez les logiciels d'enregistrement du son fourni avec votre ordinateur (disposant d'un microphone).
Pour créer des expressions parlées vous irez sur le site suivant:
http://www.research.att.com/~ttsweb/tts/demo.php.
Là, vous suivrez ces simples étapes:

* Sélectionnez une voix (hommes/femme/langue, etc…)
* tapez un texte (expression qui doit être vocalisée, deux mots de préférence). Le logiciel traduit le texte en langage parlé. Tester le fichier traduction.
* Téléchargez et enregistrer le sur votre ordinateur. Donnez lui un nom approprié, vous constaterez que le fichier est du type .wav

Il s'agit maintenant de le convertir en fichier .rso
Allez sur ce lien:
http://bricxcc.sourceforge.net/utilities.html.

* Téléchargez le programme **Wav2Rso**.
* Ouvrez Wav2Rso ainsi que votre fichier son .wav. Ce logiciel convertit ce dernier en fichier .rso, stocké dans votre ordinateur.
* Il ne reste plus qu'à ranger ce fichier .rso avec les autres dans le répertoire suivant:

1. LEGO MINDSTORMS NXT/MINDSTOMS NXT/engine/Sounds

Redémarrez le logiciel NXT-G, vous constaterez (si vous placez un bloc "Afficher" dans votre programme) l'adjonction de tous vos nouveaux sons.

Bloc "TEXTE"

Pour compléter ce tour d'horizon sur le bloc "Afficher" je voudrais ici vous présenter un nouveau bloc intitulé bloc "**Texte**". Ce bloc a cette particularité de composer des textes en phrases à partir de lettres et de mots.

Fig.12

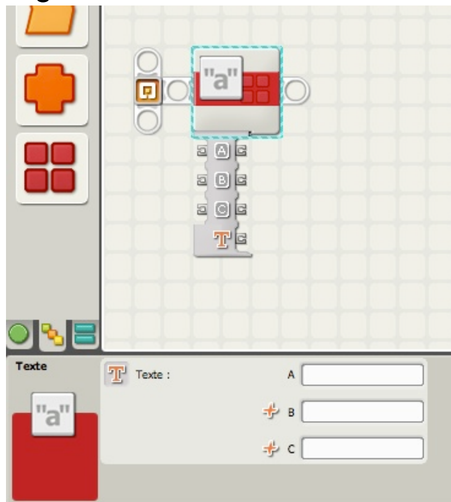
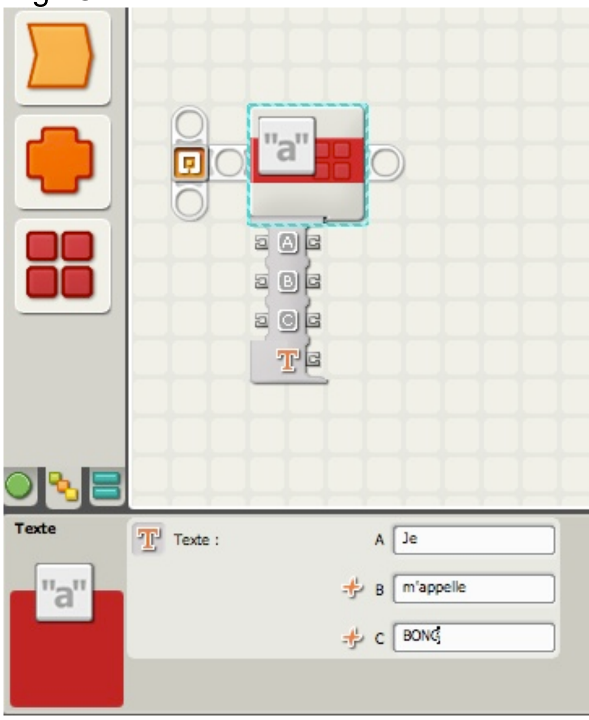


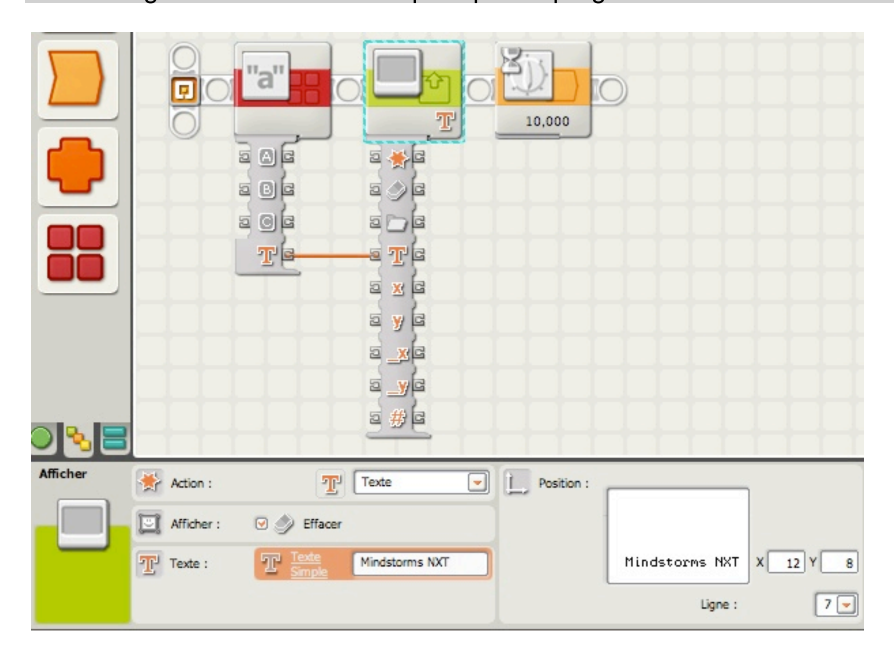
Fig.13



Il paraît tout simple, mais il est puissant.
Nous utiliserons un nouveau terme intitulé "*Chaîne*".
Une chaîne est une collection de lettres, chiffres, espaces, caractères spéciaux ou une combinaison d'entre eux. Voici quelques exemples:
* CECIESTUNECHAINE
* Qu'il en soit ainsi
* 123456789
* !&#@$%€(?:/)
* Chaque ligne est une chaîne, y compris celle ci.

Ce bloc a cette particularité de contenir 3 chaînes différentes et de les réunir en une seule, sous forme de phrase ou de longue chaîne. On appelle cela une *concaténation*.
Trois champs sont mis à votre disposition et ont comme repères A, B et C. Ces champs peuvent être saisis manuellement par vous-même, ou être remplis par l'intermédiaire des plots de saisies. Gardez en mémoire que le champ A est toujours à droite, le champ B au centre et le champ C à droite de la concaténation. Cet ordre ne peut pas être changé.
Dans l'exemple ci-dessus, A contient "Je"; B contient " m'appelle" et C contient " BONG"; ces 2 derniers précédés d'un espace. Le résultat sera "Je m'appelle BONG" quand il sera affiché sur l'écran. Et si je souhaite lire le résultat, les blocs seront assemblés ainsi:

Fig.14



Ce que nous lirons sur l'écran sera "Je m'appelle BONG" (sur la ligne 7), et non pas Mindstorms NXT.

Il découle de ceci 3 remarques:
Le bloc"Texte" ne contient que du texte (les chiffres ne sont pas considérés comme des valeurs)
le résultat, c.à d. la concaténation est un texte envoyé sur l'écran grâce à un fil de données. Ce texte remplace celui que nous voyons dans la fenêtre de visualisation.
Si la longueur du résultat dépasse la largeur de l'écran, le texte sera tronqué (il n'y a pas de passage à la ligne).

Quel intérêt, avons nous à choisir ce bloc, alors que le bloc "afficher" suffit à lui tout seul? L'avantage réside dans l'utilisation des champs A, B et C qui permettent l'affichage de textes différents, provenant eux-mêmes de blocs "Variable" et transmis par des fils de données. Dans ce cas, tout ce qui concerne l'affichage reste inchangé; seul varie le contenu. En manipulant avec adresse les variables il est possible d'afficher sur écran des messages différents associés par exemple aux capteurs.

Variables et fils de données seront abordés dans une prochaine leçon.

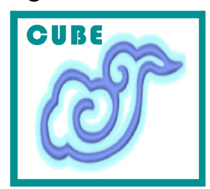


De nombreuses questions posées par les utilisateurs de NXT-G concernent les ***Plots de Données***, et cela avec raison. Les plots de données d'un Bloc peuvent semer la confusion, surtout quand le bloc présente plusieurs options. D'où certaines difficultés pour "tirer" les bons fils et pour assurer des connexions correctes. Nous allons faire une pose dans l'étude de la programmation des blocs NXT-G, et j'essaierai de vous donner quelques astuces pour mieux utiliser ces spécificités que constituent les **Plots** et les **Fils de données**.

Transmettre des informations:

Voici pour commencer un bloc de programmation factice que j'appelle **Bloc de FORME.**

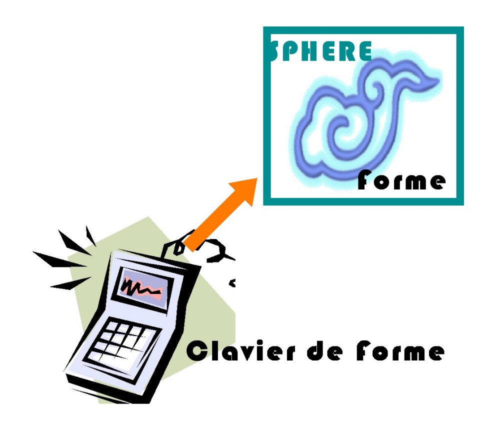
Fig.1



Ce bloc imaginaire est le plus simple que vous puissiez rencontrer. Il ne contient qu'UNE seule forme: un CUBE. Un cube et rien d'autre, et il n'existe aucun moyen pour changer cette forme.
Mais il y a encore pire, ce bloc ne dispose d'aucun dispositif pour partager cette forme avec un robot. C'est un bloc vraiment stupide, ennuyeux et inutile.

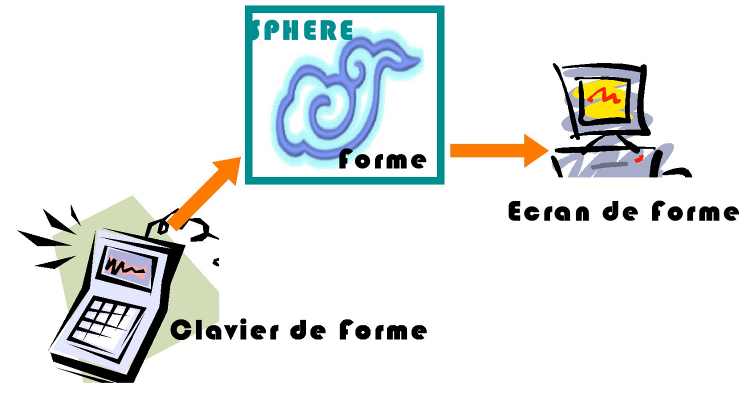
Que pourrait-on faire pour qu'il puisse nous rendre service?
Bien, d'abord le rendre capable de changer de FORME. Celle que je préfère, c'est la SPHERE. Aussi j'aimerai bien qu'il puisse remplacer le CUBE par la SPHERE.
Pour l'instant, je ne souhaite rien faire d'autre que ce changement; ce bloc contiendrait alors ma forme préférée!
La chose qui manque à ce bloc, c'est le moyen de se glisser à l'intérieur pour changer le CUBE en SPHERE. Ce serait bien si je pouvais le faire moi-même, et procéder à ces changements selon mes désirs.
La première chose à faire, c'est de connecter à ce bloc un minuscule clavier pour changer la FORME. Ce clavier très particulier comporterait des touches conçues pour choisir seulement une forme (cube, parallélépipède, sphère, cône, cylindre, anneau, tige, etc…), et rien d'autre. Il n'est pas fabriqué pour taper du texte ou autre chose.
On pourrait l'imaginer ainsi:

Fig.2



Parfait! Maintenant je peux saisir SPHERE. Plus tard, si j'envisageais de remplacer SPHERE par TIGE, je n'aurais qu'à appuyer sur la touche représentative de cette autre forme.
A présent, j'ai la forme SPHERE . En dehors de la visualiser, je ne peux pas en faire grand chose. Tout comme le clavier de forme, je vais raccorder au bloc un petit écran factice capable de puiser l'information et de l'afficher. Cet écran est semblable au clavier; il ne peut afficher que des formes et rien d'autre.
Notre dispositif pourrait ressembler à ceci:

Fig.3



Résumons:
* Le bloc ne peut contenir qu'**une seule** forme, pas de nombre, ni de jour ni de nom.
* Ensuite, au bloc FORME est raccordé un clavier qui permet de changer **seulement et uniquement** la forme contenue dans ce bloc.
* Enfin, un écran spécial raccordé au bloc, ne peut afficher que la forme, et rien d'autre; ni un nom, ni des légumes ou ingrédients…!

Et maintenant, quelques questions:
* Si je débranche le clavier, puis-je afficher la forme conservée dans le bloc? Oui, mais à condition que l'écran soit raccordé.
* Si je débranche l'écran, puis-je toujours changer la forme à l'intérieur du bloc? Oui à nouveau, mais à condition que le clavier reste raccordé.
On pourrait aussi définir le bloc de cette façon:
* Le bloc FORME ***reçoit*** une Forme comme une **ENTREE** à partir du clavier.
* Le bloc FORME ***donne*** une Forme comme une **SORTIE** sur l'écran.

Il y a dans cette description 2 termes que vous devez mémoriser: ***Entrée*** et ***Sortie*** (*input* et *output* en anglais).

Chaque fois que vous penserez bloc, rappelez vous toujours qu'une information reçue dans un bloc est une Entrée, et qu'une information donnée (ou partagée) est une Sortie.

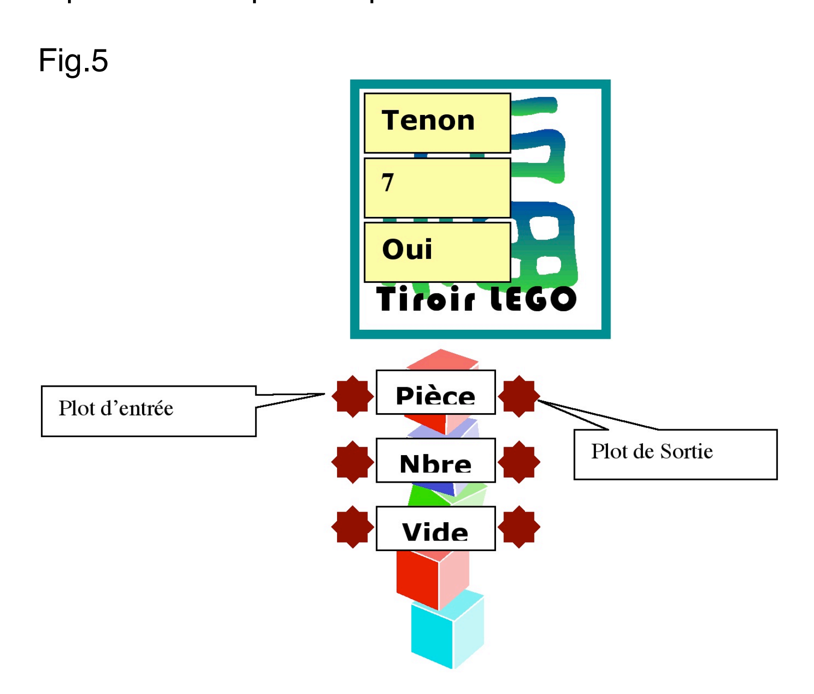
Nous allons maintenant travailler sur un bloc offrant quelques options supplémentaires. Voici un nouveau bloc factice que j'appelle TIROIR. Il est destiné à recevoir des pièces LEGO.

Fig.4



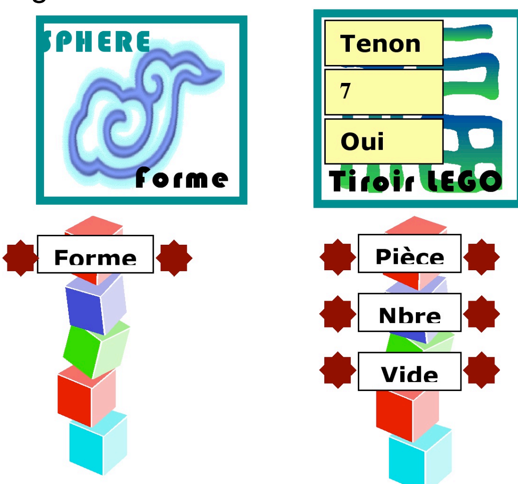
Ce bloc peut contenir 3 sortes d'informations: une forme (type de pièce), le nombre de pièces dans le tiroir,et une réponse OUI si le tiroir est vide ou NON si le tiroir n'est pas vide.
Au lieu de m'encombrer de clavier et d'écran pour les entrées et sorties, j'ai imaginé une représentation plus simple à l'aide d'un "socle" un "HUB" comme disent les anglais.

Fig.5



Vous pouvez voir sur le croquis qu'il y a 3 plots d'entrée à gauche du Hub, et 3 plots de sortie à droite. C'est sur ces plots que se raccorderont le clavier, l'écran et autres dispositifs.
Tout comme le bloc FORME, le bloc TIROIR est très pointilleux quant aux dispositifs qui lui sont raccordés. Pour le plot d'entrée PIECE, je ne peux raccorder que des formes. Nous savons déjà qu'un clavier de forme fonctionne. Je peux donc lui raccorder ce clavier et changer grâce à lui les pièces (formes), remplacer SPHERE par TENON par exemple.
Mais il y a mieux!
Rappelez-vous, le bloc FORME dont on a parlé précédemment. Bien, il a lui aussi un Hub de données, mais il était caché, escamoté. Nous allons le déployer pour le rendre visible, et pour cela, nous cliquerons au bas de l'angle gauche du bloc FORME (rappelez-vous, leçon n°2).

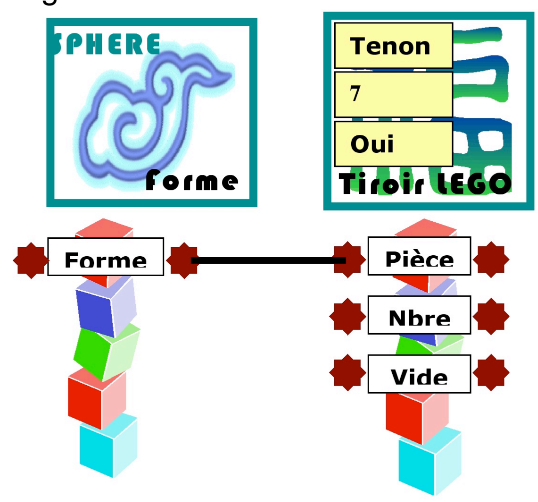
Fig.6



Le plot d'entrée à gauche du bloc FORME est la borne de raccordement du clavier qui me permet de changer la forme contenue dans le bloc. Le plot de sortie à droite peut être raccordé à un écran, mais en vérité, il peut-être raccordé à n'importe quel plot d'entrée qui accepte une forme!
Aussi, au lieu de raccorder un clavier au plot d'entrée du bloc TIROIR, j'utilise (je tire) un simple fil reliant le plot de sortie FORME du bloc FORME au plot d'entrée PIECE du bloc TIROIR.

Ouf…! J'espère que vous avez suivi.
Et voilà le résultat:

Fig.7



Je peux aussi raccorder un clavier au bloc TIROIR pour saisir un **Nombre** (une quantité de pièces). Et si j'emploie ce clavier pour autre chose que taper des valeurs, il refusera systématiquement la saisie.
Je peux aussi raccorder un clavier "logique" à l'entrée **Vide.**
Un clavier "logique" est un peu spécial: il ne peut fournir qu'UNE des 2 réponses, OUI ou NON. Pas peut-être, ni quelques fois…

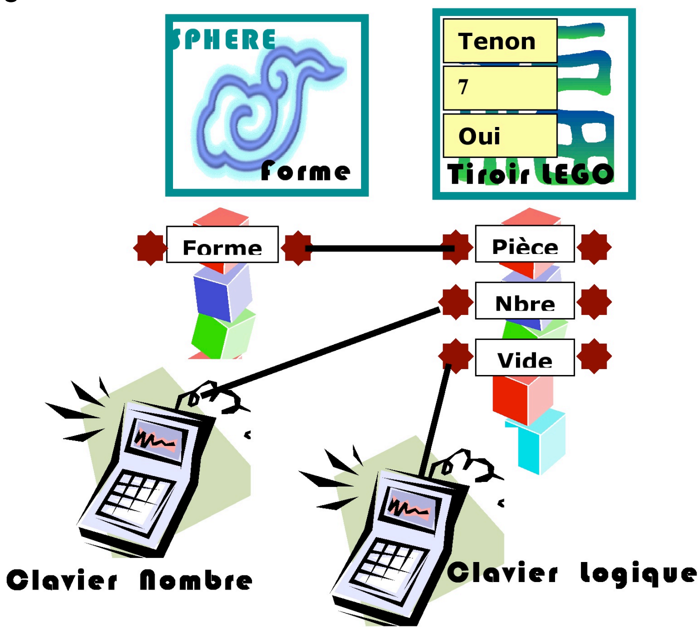
Ce que j'aimerais faire avec ce bloc TIROIR, c'est de le raccorder à un écran qui affiche une des deux choses (mais pas les deux):

* Remplir le Tiroir d'un type de pièce en précisant le nombre de pièces,

* Le tiroir n'est pas vide.

Pour réaliser cela, j'utiliserai un écran qui affichera le type de pièce et les quantités qui remplissent le tiroir. Mon dispositif ressemblerait à cela:

Fig.8



Mais, avant d'afficher le type de pièce et la quantité, j'ai besoin d'un bloc spécial, capable d'apprécier le contenu du tiroir et de signaler s'il est vide ou non. Il ne me reste plus qu'à vous révéler ce nouveau bloc que j'ai baptisé INSPECTEUR.

Le bloc INSPECTEUR est très astucieux. Il récupère une réponse OUI ou NON (logique), et selon cette réponse déclenche 2 actions différentes. L'action 1 est prise en compte si la réponse est OUI; l'action 2 si cette réponse est NON (nous verrons cela plus en détail à l'occasion d'une autre leçon).

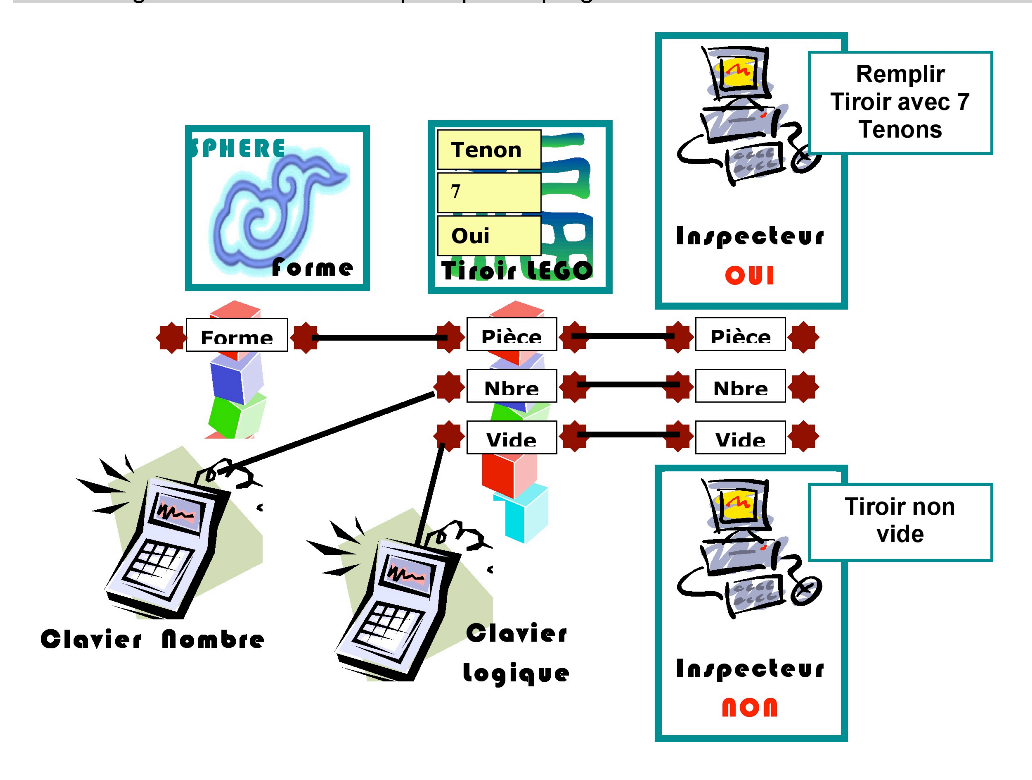
Je peux utiliser ce bloc pour évaluer le contenu du bloc TIROIR.

J'examinerai d'abord les plots de la rubrique **Vide**. Si le contenu, les informations ou tout simplement les *données* (*data* en anglais) fournies sont OUI, Le bloc INSPECTEUR confirmera un OUI sur l'écran. Si les données sont NON, l'INSPECTEUR confirmera un NON sur l'écran.

La figure ci-après illustre bien la situation quand le programme est exécuté:

L'écran affiche "Remplir Tiroir avec 7 Tenons", parce que l'inspecteur a récupéré en entrée un **OUI** provenant de la rubrique **VIDE**. *Il donne l'ordre de remplir un tiroir qui est vide.*

Fig.9



Je souhaite maintenant, changer la pièce par ROUE, et le nombre par 4. Mais aussi changer la réponse logique par NON.
L'inspecteur ayant récupéré le NON en entrée interdira le remplissage en faisant afficher "Tiroir non vide". Quelle que soit la pièce et la quantité le remplissage du tiroir n'aura pas lieu.
Pour qu'il puisse se faire, il faudra attendre qu'un OUI l'autorise, et dans ce cas, l'écran affichera: "Remplir Tiroir avec 4 Roues".

Types de données

Je pourrais créer des dizaines de blocs factices, mais j'espère que vous commencez à comprendre comment les blocs peuvent recevoir en entrée et donner en sortie des informations. Ces 2 types de données peuvent-être déterminées par vous (soit en tapant les informations, soit en les choisissant dans le panneau de configuration des blocs), ou fournies par d'autres blocs en utilisant les fils de données.
Vous devez savoir aussi, que certains blocs ne disposent pas de plots d'entrée, d'autres pas de plots de sortie, rares enfin quelques uns sans aucun plot.

Revenons à la programmation et soyez heureux d'apprendre que parmi toutes ces sortes de données imaginables, seuls trois types sont utilisés par le NXT-G:

* ***Texte***: lettres, mots, phrases et même les nombres quand ils sont considérés comme du texte.

* ***Nombre***: les nombres peuvent-être positifs ou négatifs, et quelquefois limités aux valeurs entières (seulement des nombres comme -3, 0, 4, 8 ou 10 et non pas des nombres décimaux comme 4,3 ou 8,5).
La version NXT 2.0 accepte à présent des valeurs décimales.

* ***Logique***: ne peut-être que OUI ou NON (une autre manière de l'exprimer étant **VRAI** ou **FAUX**).

Quand vous programmerez vous appliquerez cette règle:

Cinquième règle: les seules données qui peuvent circuler entre les blocs sont du TEXTE, des NOMBRES et du type LOGIQUE - et seulement ces trois.

Fils de données

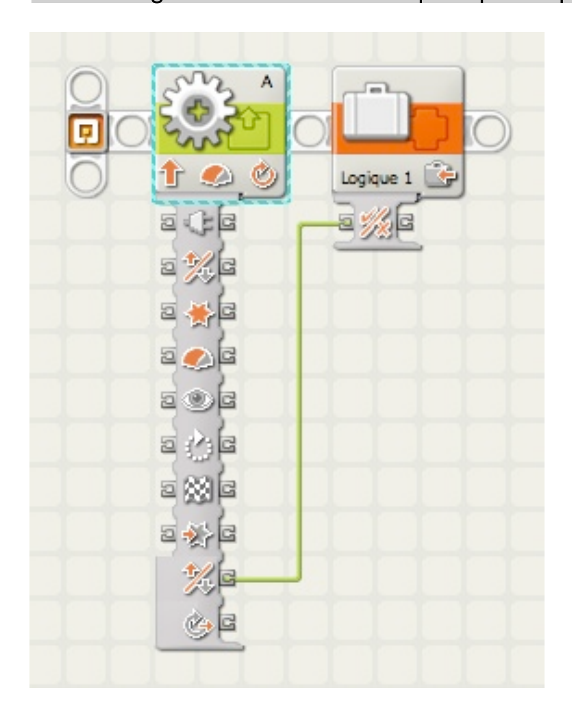
Nous avons vu que les seules données échangées entre les blocs, relèvent de trois types: Texte, Nombre et Logique.
Cela ne signifie aucunement que tous les blocs manipulent en permanence ces trois types. Certains comporteront des plots de texte et logique, mais pas de plot de nombre; d'autres n'auront pas d'entrées, mais peut-être 2 plots de sortie fournissant des données numériques. Tout dépendra des fonctions attribuées à ces blocs.

Il faudra apprendre à tirer des fils de données entre les plots sans commettre d'erreurs. Mais là aussi, tranquillisez-vous, la bonne nouvelle est que NXT-G ne vous laissera pas tirer des fils entre deux plots non compatibles (par exemple d'un plot Texte vers un plot Logique).
Il réagira par une ***rupture de fil*** et celui-ci apparaîtra en pointillé sur l'espace de travail, signifiant la rupture donc l'incompatibilité.
Mieux encore, si les liens sont correctement effectués, ils apparaîtrons sous trois couleurs distinctes, permettant de reconnaître le type de données:
* fil jaune pour une donnée numérique,
* fil orange pour une donnée textuelle
* fil vert pour une donnée logique
* Si le fil apparaît en pointillé (grisé), cela signifie qu'il est rompu et que le lien ne fonctionne pas.

Relier les plots par des fils requiert une certaine pratique. Vous constaterez qu'ils prennent parfois des directions inattendues, ou adoptent des trajets bizarres. Il faudra vous entraîner pour les maîtriser correctement.

Bien, il est temps de revenir à nos blocs NXT-G.
Nous travaillerons sur la palette "complète" de blocs

Fig.10



Le bloc MOTEUR a envoyé par ce fil une information logique sur le sens de rotation du moteur (OUI= marche avant; NON= marche arrière), à un bloc VARIABLE configuré en LOGIQUE. Cette information est stockée dans une case mémoire du bloc VARIABLE.

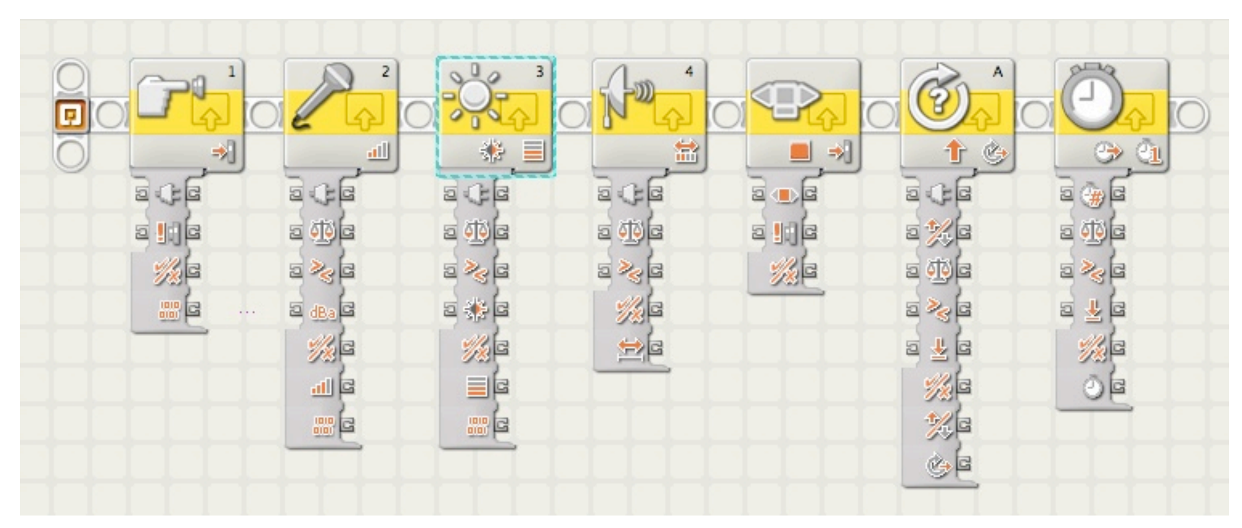
Cet exemple montre la diversité des données qu'un bloc peut manipuler.
Le Hub étant déployé, on remarque la présence des plots de part et d'autre de la "colonne" (figurés sous forme de crochet). Pour chaque entrée correspond généralement une sortie. On remarque également entre les plots une minuscule icône se rapportant à un paramètre du panneau de configuration.
Si l'on survole ces icônes avec la souris, une bulle d'aide précise encore mieux l'information liée aux données. Ces aides facilitent considérablement le travail.
Pour supprimer un fil, il suffit de cliquer sur le plot d'entrée auquel il est raccordé (le curseur prend la forme d'un doigt pointé).
Je vous conseille d'aller faire un tour dans ***l'aide et support pour Lego Mindstorms NXT***, où vous trouverez des explications détaillées sur les différents blocs et leurs plots de données. Exercez-vous à "tirer" des fils et observez leurs couleurs afin de vous familiariser avec l'interface graphique.
Une dernière bonne nouvelle: vous ne risquez pas de détruire votre programme si par hasard certains plots ne sont pas correctement reliés. Il suffit de supprimer les fils incompatibles et de recommencer.

Entrons maintenant un peu plus dans le détail.

Les capteurs:

Vous avez ici la série complète des blocs capteurs, y compris les capteurs intégrés (moteur et minuteurs). Un seul n'y figure pas, c'est le ***récepteur de message***, bloc particulier qui sera traité dans une leçon concernant Bluetooth.

Fig.11

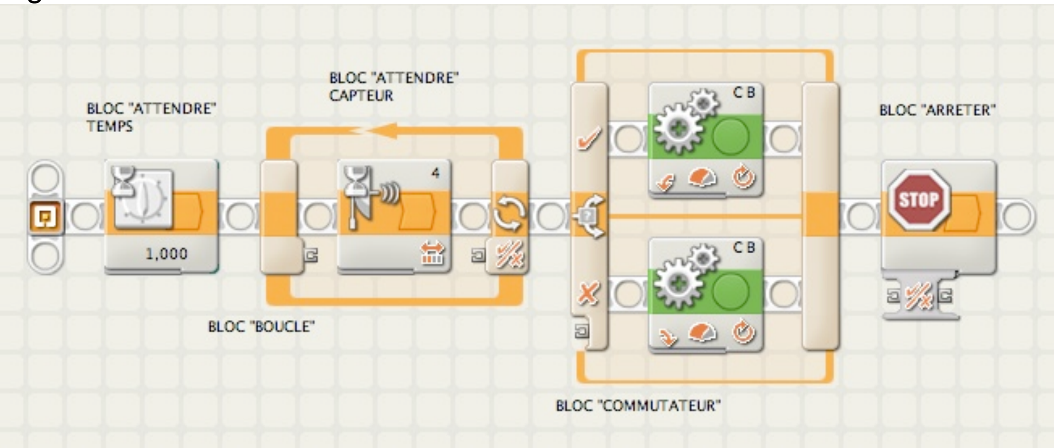


Vous remarquerez d'abord qu'il n'ont pas tous le même nombre de plots. Mais aussi que certains plots ne sont qu'en sortie seulement, signifiant que le capteur peut fournir une donnée interne.
Ils disposent tous d'une sortie logique (oui/non) signifiant qu'ils peuvent agir différemment selon certaines conditions, propres à chacun.
Un seul dispose d'une entrée/sortie numérique: le minuteur.

Les flux de données

On entend par flux les mouvements d'informations qui circulent soit à l'intérieur des blocs, soit entre blocs.

Fig.12



Comme vous le constatez, les plots sont ici peu nombreux et pour certains, il faut les débusquer.
Les plots des blocs DEPLACER ne sont pas déployés pour la circonstance.

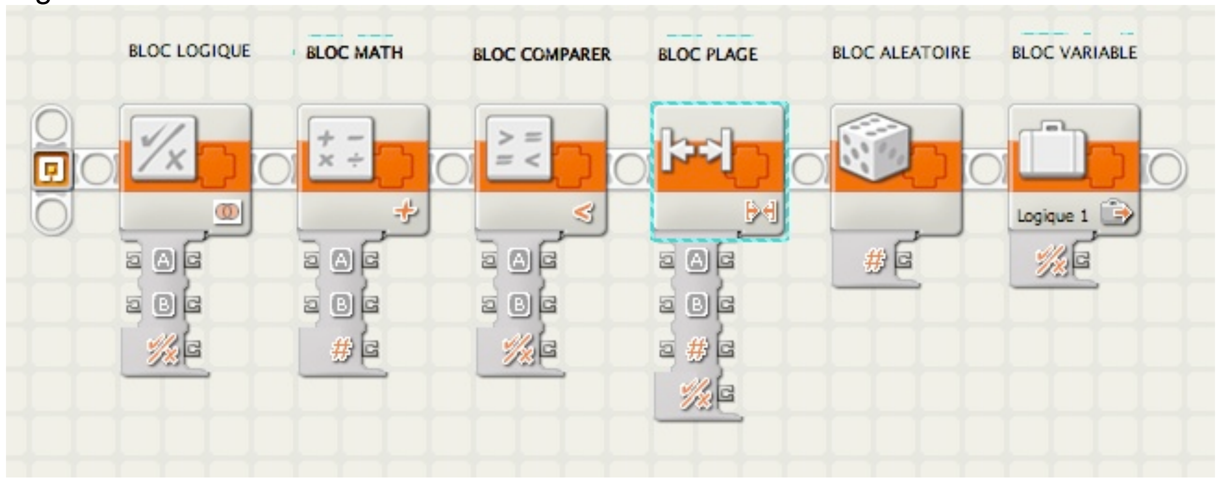
Le bloc ATTENDRE, paramétré sur TEMPS n'a aucun plot; il en est de même pour le bloc ATTENDRE, paramétré sur CAPTEUR.
Le bloc BOUCLE a un plot sortie et un plot entrée à l'intérieur de la boucle, paramétrée à partir de son panneau de configuration. Ce bloc dispose de nombreuses options (il faut cocher la case COMPTEUR pour visualiser le plot sortie). Pour le plot d'entrée, il faut choisir l'option LOGIQUE pour qu'il apparaisse.

Le bloc COMMUTATEUR n'a qu'un plot d'entrée; il est aussi paramétré à partir de son panneau de configuration.
Le bloc ARRETER n'a qu'un plot d'entrée et un de sortie.

Les Données:

Ne pas confondre avec les *flux de données*.
Ces blocs ne servent qu'à faire des calculs.

Fig.13

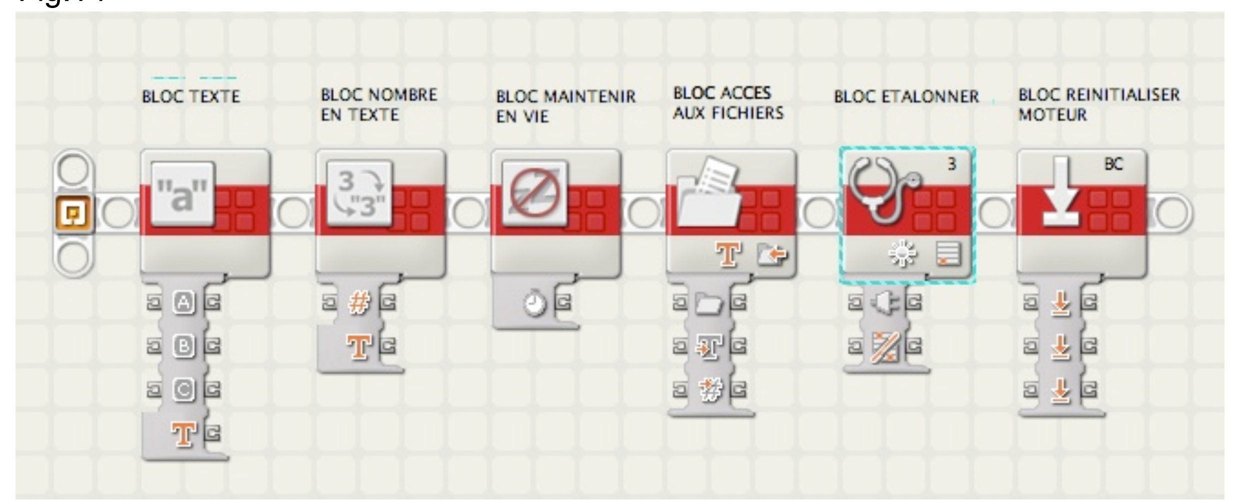


Dans les 4 premiers on observe 2 plots A et B en entrée et sortie de données.
A et B sont des cellules internes qui reçoivent et fournissent des valeurs. Ces valeurs proviennent soit d'une saisie manuelle, soit d'un autre bloc par l'intermédiaire d'un fil de données. Le résultat de l'opération est en sortie uniquement (logique, numérique).
Les 2 derniers blocs n'ont qu'une sortie uniquement.

Blocs avancés:

Ils sont donnés ici à titre d'information, à utiliser à bon escient.

Fig.14

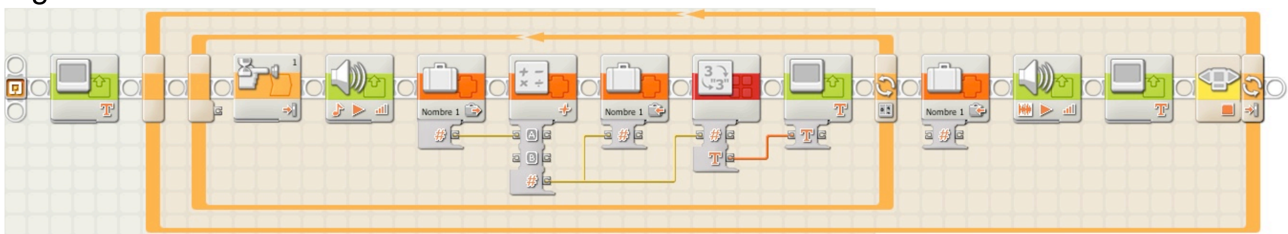


Vous connaissez déjà le premier bloc; le deuxième ne s'utilise que pour l'affichage sur écran: il transforme une valeur numérique en texte.
Les autres blocs sont surtout utilisés par les programmeurs confirmés.

Exemple d'utilité des fils de données

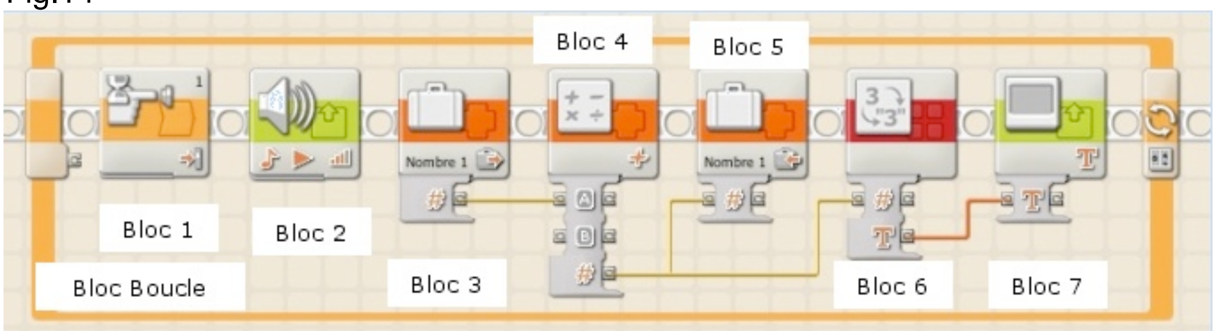
Je souhaite créer un compteur qui à l'aide du capteur tactile, enregistre chaque coup de bouton et affiche sur écran son numéro d'ordre. Pour compliquer un peu plus, je souhaite que ce compteur revienne à zéro, chaque fois que l'on dépasse une valeur plafond.
Pour vous faire une idée, voici à quoi ressemble ce programme:

Fig.15



Nous allons nous intéresser à la boucle intérieure:

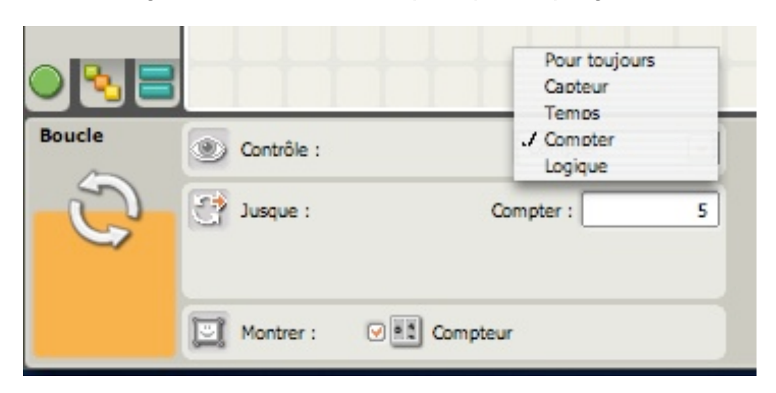
Fig.14



Bloc 1: capteur tactile, chaque fois qu'il est activé, il émet un top sonore (bloc 2).
Bloc 3: variable numérique initialisée à la valeur zéro, valeur de départ. dans ce bloc on ne fait que lire le contenu de la variable.
Bloc 4: Math, option pour effectuer une addition. Ce bloc comporte 2 cases A et B. Je tire un fil entre Bloc 3 et A (pour mettre dans A la valeur écrite dans le bloc, c. à d. zéro). Dans B j'écris une valeur +1. Le résultat (#) devient A + B = 1.
Bloc 5: on écrit dans la variable la nouvelle valeur qui devient 1, en tirant un fil entre les plots # des blocs 4 et 5.
Bloc 6: on transforme la valeur numérique en texte pour l'afficher dans le bloc 7. Un fil est tiré du plot # bloc 4 vers plot # bloc 6.
Bloc 7: On tire un fil entre plot T bloc 6 vers plot T bloc 7 qui affiche 1.
Cette boucle vient d'être exécutée une fois. Elle va se répéter plusieurs fois, très exactement 4 fois puisque je l'ai décidé ainsi.

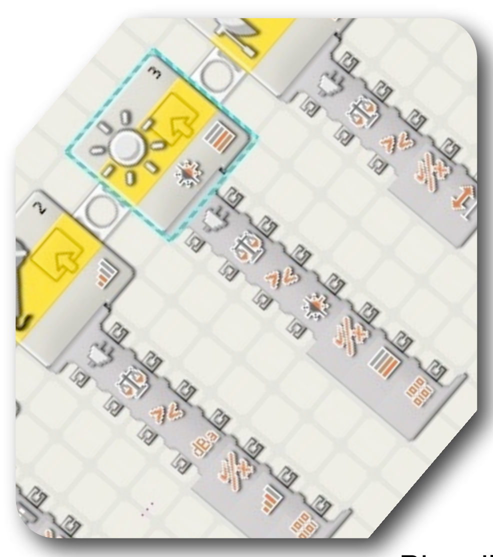
Examinons maintenant le panneau de configuration de la boucle.

Fig.15



Dans la zone CONTROLE, le champ est configuré sur *Compter* (sous-entendu compter le nombre de cycles). Ce nombre a été fixé à 5. Notez également que la case compteur est cochée, ce qui permet d'afficher le plot de sortie de la boucle (ici inemployée), donc de récupérer éventuellement cette valeur.
La boucle va donc se répéter 4 fois, et à la 5 ème, la boucle se termine.

Le programme se poursuit dans la boucle extérieure.
On écrit la valeur zéro dans le bloc variable, suivit d'un top sonore différent du premier pour dire qu'on recommence à partir de zéro.
Enfin pour quitter le programme, on appuie à ce moment sur le bouton rouge du NXT.



Leçon n°5: OUI ou NON?... Question pour les capteurs

Quelle différence trouvez-vous entre ces deux questions?
- Quel temps fait-il?
- Pleut-il?

Bien, il y a pas mal de différences: la première contient 3 mots, et la seconde un seul, par exemple.
Mais la véritable raison, c'est d'attirer votre attention sur la première car elle est ce qu'on appelle une question ouverte; Il peut faire beau, venteux, brumeux ou toutes autres sortes de réponses.
La seconde question n'a que deux réponses possibles: Oui ou Non.
Lorsque vous programmez votre robot, vous devez comprendre que la plupart du temps, il ne peut vous renseigner que par ces deux réponses: Oui ou Non. En d'autres circonstances, il ne peut *comprendre* seulement qu'un réponse Oui ou Non. Et cette façon de renseigner ou de comprendre se fera par les capteurs qui sont justement conçus pour réagir dans ce sens.
C'est ce que je vais essayer d'expliquer au cours de cette leçon.

Mais d'abord,
rappel des règles
*** lère règle: familiarisez-vous avec les outils de programmation.**
*** 2ème règle: les capteurs doivent être connectés aux ports d'entrée (1,2,3 & 4) et les moteurs doivent être connectés aux ports de sortie (A, B & C).**
*** 3ème règle: Un bloc ne peut accomplir qu'une seule tâche à la fois. Il faut autant de blocs que de tâches à accomplir.**
*** 4ème règle: un programme NXT-G se termine une fois le dernier bloc exécuté.**
*** 5ème règle: les seules données qui peuvent circuler entre les blocs sont du TEXTE, des NOMBRES et du type LOGIQUE - et seulement ces trois.**

L'un ou l'autre.

Reprenons notre travail avec BONG et jouons à nouveau au jeu des questions/réponses.
LEO > BONG, quelle est la couleur du mur en face?
(BONG ne réagit pas et reste immobile…)
LEO > BONG, quelle est la position du bouton de votre capteur tactile?
(BONG ne réagit pas et reste immobile…)

Humm… BONG me semble peu réceptif aujourd'hui! Je crois me souvenir que BONG préfère répondre aux questions par Oui ou par Non. Essayons autrement.
LEO > BONG, le mur d'en face est-il bleu?
(BONG le mot "OUI" apparaît sur son écran…)

LEO > BONG, Le bouton du capteur tactile est-il appuyé?
(BONG le mot "NON" apparaît sur son écran…)

Bon, j'ai l'impression d'avancer. BONG préfère communiquer avec moi par des Oui et des Non en réponses à mes questions.
Une autre façon de dire cela, est que BONG préfère communiquer à l'aide de *réponses logiques*: une réponse logique est tout simplement Oui ou Non.

Poursuivons notre conversation:
LEO > BONG, votre capteur à UltraSons, a-t-il détecté un objet à 20 cm en face de vous?
(BONG > Vrai).
LEO > BONG, votre bouton flèche droite set-il appuyé?
(BONG > Faux).

Apparemment, les capteurs de BONG savent envoyer une réponse à BONG qui peut à son tour me la transmettre. BONG "écoute" les réactions des capteurs, et répond par Vrai ou Faux.
En quoi tout cela concerne-t-il la programmation?
Et bien, retenez ceci: votre robot NXT peut envoyer et recevoir des réponses logiques à partir des capteurs, moteurs, boutons et autres dispositifs.

Voici un exemple:

Fig.1

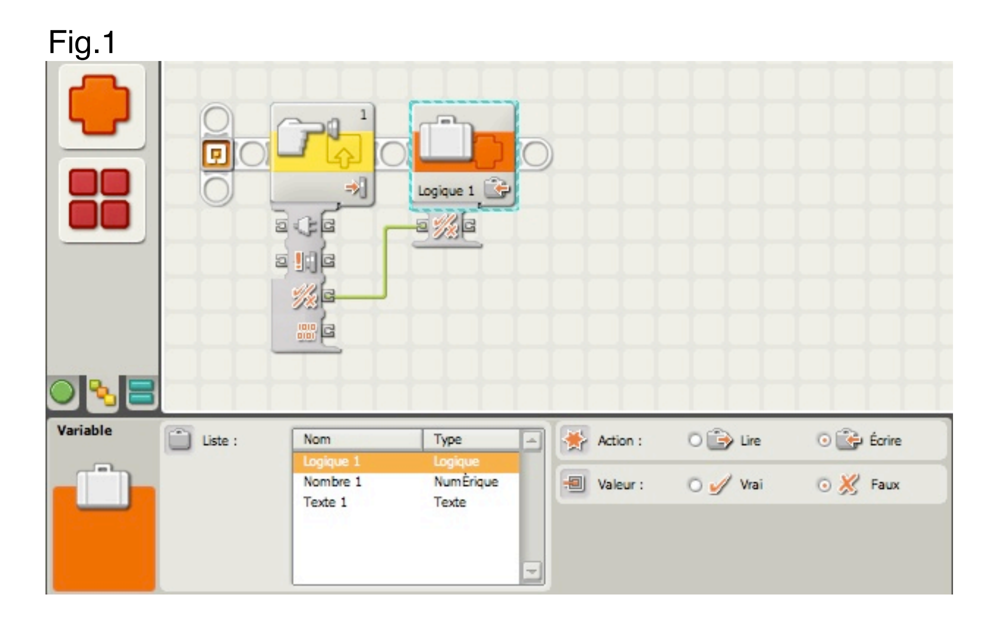
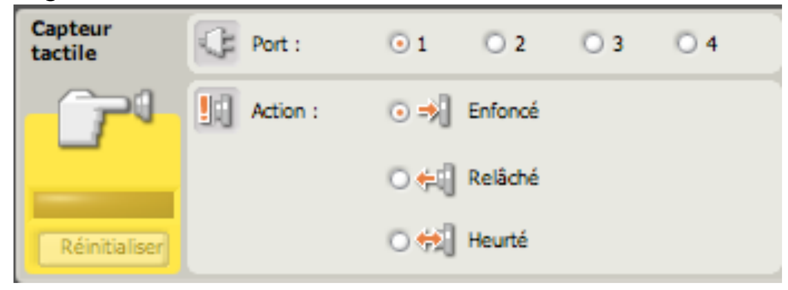



Fig.2



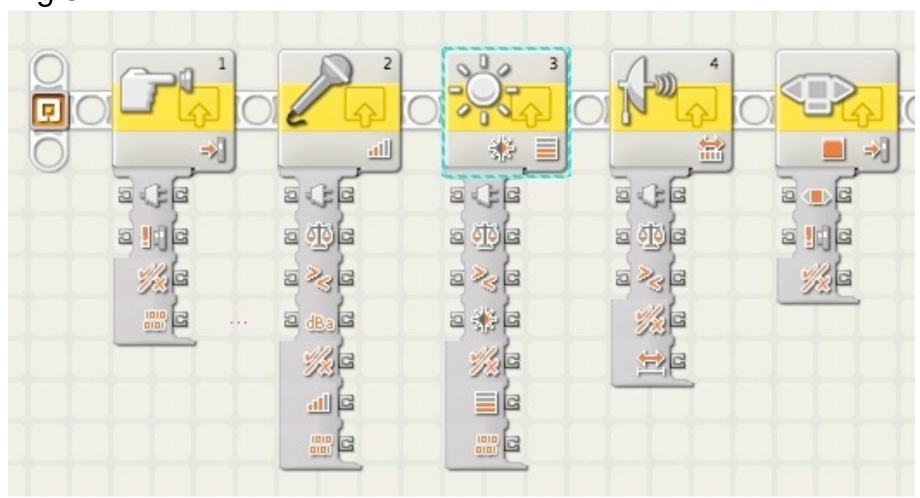
Le bloc **Capteur tactile** que vous apercevez a son Hub déployé. Lorsque vous survolez à l'aide de la souris la petite icône ***v/x,*** vous voyez apparaître une "bulle" d'aide ***Oui/Non.***

Cela vous informe que le plot de sortie peut, par le moyen d'un fil de données fournir une donnée de *type logique* à un autre bloc de programmation.
Mais, comment savoir si la sortie est un Oui ou un Non?
Facile, il suffit d'examiner le panneau de configuration du bloc Capteur tactile. La réponse réside dans le choix que vous ferez dans la zone "*action*". Ici, "*Enfoncé*" est sélectionné. Donc si le bouton est enfoncé, il enverra au bloc voisin une valeur logique "Oui" ou "Vrai" (les 2 termes ayant souvent la même signification).
Le bloc voisin est un **bloc Variable** (que nous étudierons plus tard) dont le type est logique; il est paramétré pour écrire une donnée logique dans sa mémoire (avec une valeur de défaut "Faux" identique à "Non").
Tant que le bouton du capteur tactile est relâché (valeur Faux), le bloc Variable garde dans sa mémoire une donnée logique "Faux".
Quand le capteur tactile est appuyé, il envoie une valeur "Vrai" au bloc Variable, qui reçoit par le canal du fil de données cette valeur "Vrai". Cette nouvelle valeur remplace la précédente et elle est ensuite stockée dans la mémoire du bloc Variable pour un autre usage.

Rappelez-vous, quand vous connectez des blocs de programmation par des fils de données, ces fils ne seront effectifs que s'ils relient *impérativement* des plots de même type (Logique, Texte, Nombre).
J'attire votre attention sur certains blocs qui par défaut contiennent une valeur Vrai ou Faux. Cette valeur peut être inversée par vos soins pour les besoins de vos programmes. Dans le cas de notre exemple, elle n'a pas été modifiée.

Le type de données *Logique* se retrouve dans plusieurs blocs, et en particulier dans les blocs de **capteurs**.


Fig.3



C'est dans l'utilisation des blocs **Boucle** et **Commutateur** que les données du *type Logique* affirment leur puissance. Elles sont très utiles quand un robot doit décider une action de sa propre initiative. Grâce à ces blocs, il peut développer des comportements complexes avec un meilleur contrôle de lui-même.
Le robot peut scruter des capteurs, moteurs ou autres sources d'entrées, et en fonction des réponses Oui/Non, prendre des décisions pour les actions suivantes.

Réactions

Votre kit Mindstorms NXT comporte une collection de moteurs et de capteurs; de plus, la brique NXT dispose aussi de boutons et de 3 minuteurs. Seriez-vous surpris d'apprendre

que ces éléments sont susceptibles de fournir des informations à la brique, simplement par réaction?
Pour les capteurs, cela est évident puisqu'ils sont conçus pour répondre à des sollicitations externes, comme la lumière, le contact ou le son, et transmettent ces réactions au NXT. Mais, qu'en est-il des moteurs? Et comment un minuteur peut-il être utilisé comme entrée ou communiquer ses réactions à la brique? C'est ce que je tenterai d'expliquer dans ce qui suit.

Quelles sont vos conditions?

Nous commencerons avec ce mot: "***CONDITION***".
Prenons un exemple: Les feux de circulation automobiles sont soumis à *trois conditions*: ils peuvent-être rouge, jaune ou vert. Un interrupteur électrique ne comporte que *deux conditions* de fonctionnement: allumé ou éteint. A partir de ces deux exemples, nous allons soumettre à BONG quelques pseudo-codes.

LEO > BONG, avance jusqu'à ce que les feux passent au rouge.
LEO > BONG, affiche le mot "ALLUME" sur ton écran, jusqu'à ce que l'interrupteur soit en position *éteint*.
Dans ces deux cas, je présume que BONG a des yeux et qu'il peut voir les feux ou l'interrupteur. Si BONG n'a pas d'yeux, il faut alors trouver un moyen pour que les feux de trafic ou l'interrupteur transmettent leur état à BONG. Ils pourraient alors communiquer leurs réactions, ou leurs entrées, à BONG.
Demander à des feux de trafic de fournir une *entrée*, n'est pas réaliste, mais, puisque le NXT est en mesure de recevoir des informations à partir des capteurs, modifions les pseudo-codes:

LEO > BONG, avance tant que le capteur photosensible renvoie la valeur 20.
LEO > BONG, affiche le mot "Salut" jusqu'à ce que le capteur tactile signale qu'il a été appuyé puis relâché.
LEO > BONG, joue la note DO quand le capteur à ultrasons détecte un objet situé à 20 cm devant toi.

Les moteurs peuvent, eux aussi fournir des informations sur leur état.
LEO > BONG, fait tourner le moteur A jusqu'à ce que le moteur B signale une rotation de 10 degrés.
LEO > BONG, affiche "5 rotations" quand le moteur C a tourné 5 fois.

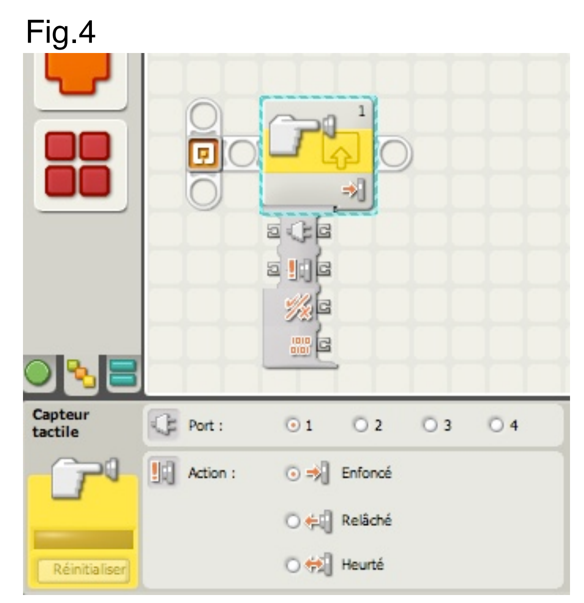
J'ai signalé plus haut que le NXT disposait de 3 minuteurs internes, plus 3 boutons. Je pourrais aussi écrire ce pseudo-code:
LEO > BONG, lorsque 20 secondes seront écoulées, tourne de 90 degrés.
Ou encore,
LEO > BONG, joue la note SI quand j'appuie sur le bouton flèche gauche, et la note DO quand j'appuie sur le bouton flèche droite.
Bien, vous constatez que les capteurs, moteurs, boutons et minuteurs, peuvent fournir des données en entrées au NXT pour contrôler d'autres actions (DEPLACER, SON, et autres blocs).
Lorsque vous programmerez vos robots ils devront accomplir des actions basées sur les conditions de ces dispositifs. Tout comme un interrupteur qui ne connaît que 2 états, vous devez connaître les différentes conditions liées aux capteurs, moteurs, boutons et minuteurs et qui peuvent-être transmises au NXT.

Paramétrer les capteurs.

Quelques remarques préliminaires:
En premier, les capteurs détectent le changement d'une *condition* qui peut-être le niveau de luminosité, le volume d'un son, la position (mouvement). Le capteur ne fait qu'attendre ce changement. Le capteur photosensible , par exemple, peut détecter un changement du niveau d'éclairement d'une chambre.
Deuxièmement, les blocs de programmation ne peuvent *seulement* répondre qu'à *une condition à la fois*. Un capteur photosensible, par exemple, ne peut-être configuré pour répondre à une valeur en-dessous de 70 et au-dessus de 30. Pour réaliser ces 2 conditions, il faudra utiliser 2 blocs *capteur photosensible* dans votre programme.
En fait, les capteurs ne fournissent qu'un type de réponse logique: Vrai ou Faux.
Dans l'exemple du trafic autos, on peut supposer que chaque véhicule dispose d'un capteur fixé sur son toit. Il est programmé pour réagir aux couleurs des feux. Les véhicules se déplaceront si la couleur est au vert et s'arrêteront au rouge. Ils ralentiront si le feu passe au jaune.
Ces capteurs ne devront détecter qu'une des 3 conditions: vert, rouge ou jaune. Un capteur NXT ne pouvant répondre qu'à une condition à la fois, il faudra donc utiliser 3 capteurs:
* si le 1er capteur détecte le vert, le véhicule continuera son avancée,
* si le 2ème capteur détecte le jaune, le véhicule ralentira,
* si le 3ème capteur détecte le rouge, le véhicule s'arrêtera.
A remarquer qu'une seule condition ne peut-être Vrai à un moment donné; les feux ne pouvant être en même temps vert et rouge par exemple. Il faudra donc procéder par paire en éliminant les incompatibilités (rouge et jaune ne peuvent pas provoquer un déclenchement d'action).

Capteur tactile

Fig.4



Notez que le port de connexion est le 1 par défaut. Mais rien ne vous empêche d'en choisir un autre.

Si vous acceptez le choix de défaut pour tous les capteurs, vous pourrez alors vous constituer une petite bibliothèque de programmes réutilisables quels que soient les robots conçus(à condition qu'ils soient tous câblés de la même manière).
Dans la zone *Action*, 3 trois cas sont envisagées; à remarquer que le choix *Heurté* signifie Enfoncé et Relâché rapidement (moins d'une demie seconde entre les deux). Ce choix est très utile quand il s'agit de compter par exemple des impulsions.
Enfin, n'oubliez pas qu'en survolant à l'aide de la souris sur les plots, vous affichez une "bulle" précisant le type de donnée.

Fig.5

Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Capteur tactile :

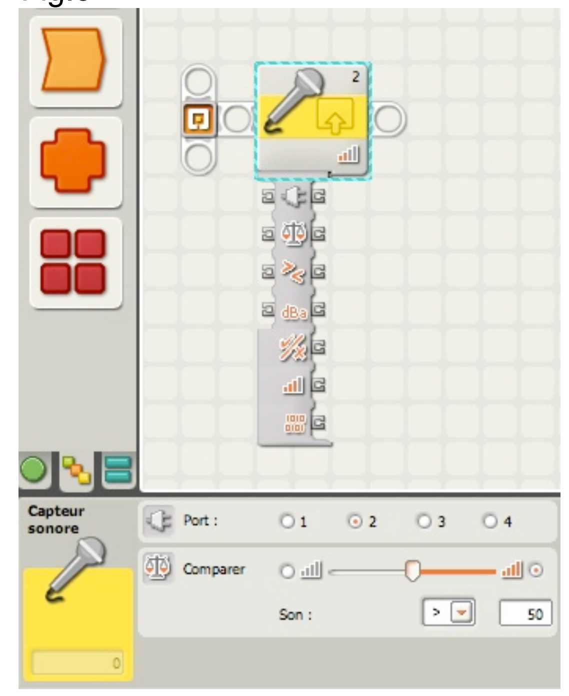
	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Port	Numérique	1 - 4	1 = Port 1, 2 = Port 2, 3 = Port 3, 4 = Port 4	
	Action	Numérique	0 - 2	0 = Enfoncé 1 = Relâché 2 = Heurté	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Valeur brute	Numérique	0 - 1024	Valeur brute (sans mise à l'échelle) lue depuis le capteur	

Ce tableau est extrait du Menu Aide du logiciel NXT-G et il résume les valeurs transitant par les plots reliés par des fils de données.

Capteur Sonore

Ce bloc est un détecteur de sons. Il peut, via des fils de données, envoyer le niveau sonore actuel, ainsi qu'un signal logique (vrai/faux) qui varie selon que le niveau sonore actuel est supérieur ou inférieur à un *point de déclenchement.*

Fig.6



Le port de connexion de défaut est le 2.

Le *point de déclenchement* est la valeur précise, dans une plage de nombres, où se produit un changement de conditions. Par exemple, vous pouvez programmer votre robot pour qu'il n'avance que si le niveau sonore dépasse 40%. Le point de déclenchement vaut dans ce cas 40.
Dans la zone *Comparer*, vous choisissez une plage de valeurs de 1 à 100 à l'aide de la glissière, ou en entrant une valeur dans la zone de saisie.
Pour définir la plage (au-dessus ou en dessous du point de déclenchement) qui générera le signal « vrai », utilisez les cases d'option ou le menu déroulant. La partie « vrai » de la plage est colorée et la partie « faux » est grisée.
Par défaut, le bloc Capteur sonore est paramétré pour qu'un niveau sonore supérieur à 50% génère un signal « vrai ». Le bouton d'option à droite de la plage est activée et la glissière est réglée sur 50. Pour activer les parties « vraies » de la plage (définir les valeurs sous 50% comme étant « vraies »), activez le bouton d'option de gauche.

Astuce: réglage du point de déclenchement pour un capteur sonore
si vous voulez qu'un applaudissement sonore génère le signal « vrai », applaudissez à plusieurs reprises et observez les valeurs de niveau sonore affichées dans la zone jaune de rétroaction. Si vos applaudissement génèrent des niveaux sonores d'environ 80%, fixez la valeur de déclenchement à 70. Ce faisant, seuls les niveaux sonores au-delà de 70% (comme vos applaudissements) génèreront un signal « vrai ». Les sons moins forts seront ignorés.

Le point de déclenchement peut également être défini de manière dynamique via un fil de données en entrée.

Fig.7

Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Capteur sonore :

	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Port	Numérique	1 - 4	1 = Port 1, 2 = Port 2, 3 = Port 3, 4 = Port 4	
	Point de déclenchement	Numérique	0 - 100	Valeur de référence de la comparaison	
	Supérieur/Inférieur	Logique	Vrai/Faux	Logique utilisée dans la comparaison : Vrai = Supérieur, Faux = Inférieur	
	dBA	Logique	Vrai/Faux	Vrai = Mode dBA Faux = Mode dB	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Niveau sonore	Numérique	0 - 100	Valeur d'échelle lue depuis le capteur	
	Valeur brute	Numérique	0 - 1024	Valeur brute (sans mise à l'échelle) lue depuis le capteur	

Le capteur sonore offre à votre robot la faculté d' « entendre » des sons. En vérité il n'entend pas comme un être vivant, mais il est capable de percevoir une pression acoustique, qui se traduit par une vibration sonore à un niveau plus ou moins fort. Cette pression est exprimées en *décibels*, unité de mesure.

Il existe deux types de décibels :

* Le décibel ajusté (en abrégé dBA) qui exprime une sensibilité adaptée à celle de l'oreille humaine. C'est celle que vos oreilles peuvent entendre.

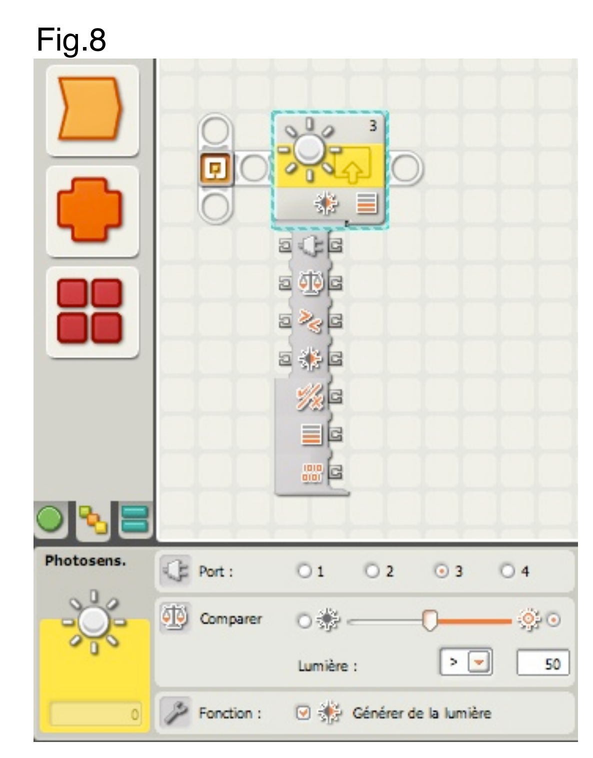
* Le décibel standard (en abrégé dB) adapté à tous les sons, y compris ceux que l'oreille humaine ne peut pas entendre. On peut citer par exemple les sons à fréquences élevées auxquels les chiens sont sensibles. Tous les sons, dans ce cas, sont mesurés d'une manière identique.

La mesure des niveaux de pression étant d'une grande complexité qu'il est impossible d'aborder ici, les valeurs du capteur sonore sont affichés sur l'écran du NXT en pourcentages (%). A titre d'exemple, 4-5% correspond à un local silencieux. Un relevé de 5-10% s'obtient si une personne parle à une certaine distance de l'appareil. 10-30% correspond à une conversation normale à proximité du capteur, ou à de la musique d'ambiance à volume normal. Enfin, on obtient des relevés de 30-100% si des personnes crient ou si de la musique est diffusée à un volume élevé.

Capteur Photosensible

Ce capteur détecte la lumière ambiante (avoisinante). Il peut, via des fils de données, envoyer la valeur actuelle correspondant à la luminosité mesurée, ainsi qu'un signal logique (vrai/faux) qui varie selon que la valeur actuelle de la luminosité est supérieure ou inférieure à un *point de déclenchement*.

Fig.8



Le port de connexion de défaut est le 3.

Le *point de déclenchement* est la valeur précise, dans une plage de nombres, où se produit un changement de conditions. Par exemple, vous pouvez programmer votre robot pour qu'il n'avance que si la luminosité dépasse 60%. Le point de déclenchement vaut dans ce cas 60.

La zone Comparer s'utilise comme celle du capteur sonore.

Astuce: réglage du point de déclenchement pour un capteur photosensible
Si vous voulez qu'une torche dirigée vers votre robot génère le signal « vrai », dirigez la torche vers le robot à plusieurs reprises et observez les valeurs de luminosité affichées dans la zone jaune de rétroaction. Si la torche génère des niveaux de luminosité d'environ 80%, fixez la valeur de déclenchement à 70. Ce faisant, toute lumière au-delà de 70% (comme celle de la torche) génèrera un signal « vrai ». La lumière plus faible sera ignorée.

Si vous activez la case à cocher « *Lumière générée* », le capteur photosensible allume sa petite source lumineuse et détecte cette lumière si elle est réfléchie dans sa direction. Cette fonction est particulièrement utile dans des conditions lumineuses difficiles, comme une pièce très éclairée. Cette fonction permet au capteur photosensible de faire office de télémètre à courte portée. Lorsque l'option « *Lumière générée* » est activée, le capteur photosensible détecte un niveau de lumière réfléchie plus élevé s'il approche d'un objet réfléchissant. Il est dès lors possible d'utiliser ce capteur pour éviter de heurter des obstacles.

Le point de déclenchement peut également être défini de manière dynamique à l'aide d'un fil de données en entrée.

Fig.9

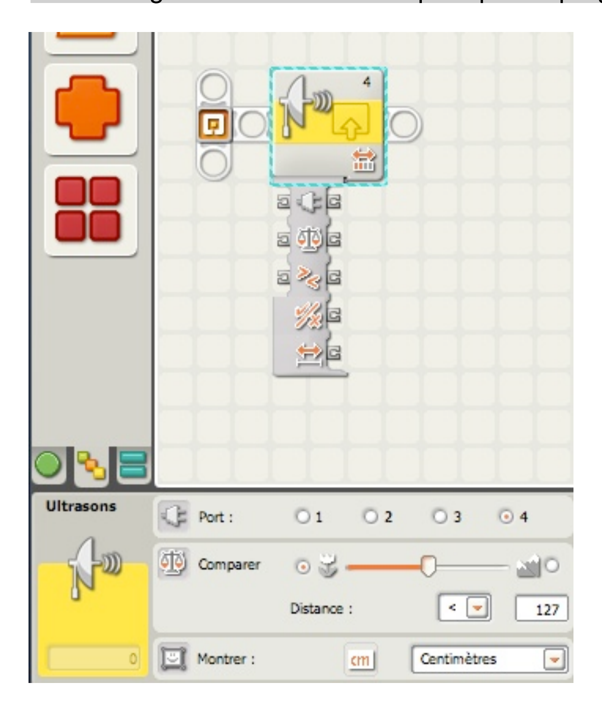
Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Capteur photosensible :

	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Port	Numérique	1 - 4	1 = Port 1, 2 = Port 2, 3 = Port 3, 4 = Port 4	
	Point de déclenchement	Numérique	0 - 100	Valeur de référence de la comparaison	
	Supérieur/Inférieur	Logique	Vrai/Faux	Logique utilisée dans la comparaison. Vrai = Supérieur, Faux = Inférieur	
	Générer de la lumière	Logique	Vrai/Faux	Détermine si la LED du capteur est allumée ou éteinte.	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Intensité	Numérique	0 - 100	Valeur d'échelle lue depuis le capteur	
	Valeur brute	Numérique	0 - 1024	Valeur brute (sans mise à l'échelle) lue depuis le capteur	

Capteur à Ultrasons

Ce bloc peut détecter des objets distants, au maximum, d'environ 250 cm. Il peut, via des fils de données, envoyer la valeur actuelle correspondant au relevé ultrasonique actuel, ainsi qu'un signal logique (vrai/faux) qui varie selon que le relevé ultrasonique actuel est supérieur ou inférieur à un *point de déclenchement*.

Fig.10



Le port de connexion de défaut est le 4.

Le *point de déclenchement* est la valeur précise, dans une plage de nombres, où se produit un changement de conditions. Par exemple, vous pouvez programmer votre robot pour qu'il attaque quand un objet s'approche de moins de 50 cm de la portée maximale du capteur d'ultrasons. Le point de déclenchement vaut dans ce cas 50.
La zone Comparer s'utilise comme celle du capteur sonore.

Astuce: réglage du point de déclenchement pour un capteur d'ultrasons
Si vous voulez qu'un intrus approchant de votre robot génère le signal « vrai », orientez le capteur d'ultrasons vers l'intrus (ou tout objet possédant la même réflectivité) en plaçant celui-ci à différentes distances du capteur. Observez les relevés ultrasoniques affichés dans la zone jaune de rétroaction. Si l'intrus génère un relevé ultrasonique d'environ 127 cm à la distance la plus proche selon laquelle vous voulez l'autoriser à approcher du robot, fixez la valeur de déclenchement à 127. De la sorte, si un objet possédant la même réflectivité s'approche de plus de 127 cm, le bloc Capteur d'ultrasons génère un signal « Vrai ». En connectant un fil de données à un bloc Déplacer ou Moteur, vous pouvez ordonner à votre robot d'attaquer ou de reculer.

Les valeurs peuvent être affichées en centimètres ou en pouces.
Si vous activez le bouton d'option à gauche de la glissière, le bloc sera déclenché lorsqu'il détecte un objet plus proche que la distance de déclenchement ; activez le bouton d'option à droite pour déclencher le bloc lorsqu'il détecte un objet plus éloigné que cette distance. Utilisez la glissière pour définir la distance de déclenchement, ou entrez directement une valeur dans la zone de saisie (0 à 250 si le bloc est configuré en centimètres et 0 à 100 s'il est configuré en pouces).
La plage de détection se situe entre 0 (distance la plus proche) et 255 cm (distance la plus éloignée) environ (précision +/- 3 cm).
N'oubliez pas que les surfaces très réfléchissantes peuvent être détectées à de plus grandes distances que les surfaces non réfléchissantes.

Remarque : si plusieurs capteurs d'ultrasons fonctionnent dans la même pièce, il se peut qu'ils perturbent leurs relevés respectifs.

Vous pouvez contrôler le bloc Capteur d'ultrasons de manière dynamique en connectant à son plot de données des fils de données (provenant des plots de données d'autres blocs).

Fig.11

Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Capteur d'ultrasons :

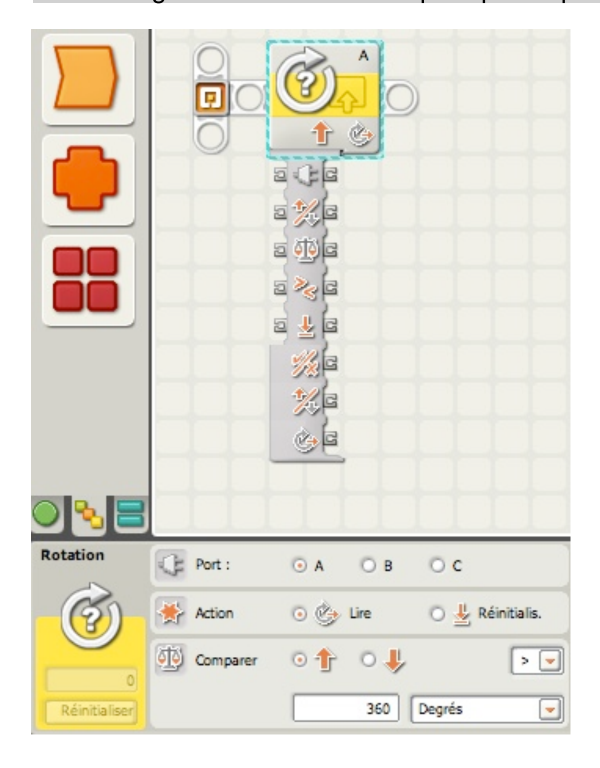
	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Port	Numérique	1 - 4	1 = Port 1, 2 = Port 2, 3 = Port 3, 4 = Port 4	
	Point de déclenchement	Numérique	0 - 255 (cm) 0 - 100 (pouces)	Valeur de référence de la comparaison	
	Supérieur/Inférieur	Logique	Vrai/Faux	Logique utilisée dans la comparaison : Vrai = Supérieur, Faux = Inférieur	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Distance	Numérique	0 - 255 (cm) 0 - 100 (pouces)	Valeur d'échelle lue depuis le capteur	

Nous allons à présent aborder les capteurs dit "intégrés", c'est à dire existant par construction dans les moteurs ou dans le NXT. Inutile de cherchez dans le kit # 8527, ils n'existent pas sous la même forme que ceux déjà vus.
Pour y accéder, il n'y a qu'un seul moyen, c'est par celui des blocs de programmation.

Bloc Capteur de rotation

Ce bloc compte le nombre de degrés (une rotation complète vaut 360 degrés) ou de rotations complètes selon lequel votre moteur tourne. Ce bloc peut envoyer, via des fils de données, le nombre actuel de degrés ou de rotations, ainsi qu'un signal logique (vrai/faux) qui varie selon que le nombre de degrés ou de rotations est supérieur ou inférieur à un *point de déclenchement.*

Fig.12



Zone *Port*: choisir le port que le bloc doit surveiller (A, B ou C).
Zone *Action*: Indiquer s'il s'agit de lire la valeur actuelle d'un capteur de rotation ou remettre la valeur du capteur à zéro.
Zone *Comparer*: définir la direction à surveiller à l'aide des cases d'option : vers l'avant ou vers l'arrière.

Important : veillez à choisir la direction correcte, faute de quoi le bloc sera incapable de compter jusqu'à la valeur de déclenchement.

Utiliser le menu déroulant pour indiquer si le bloc doit compter le nombre de rotations ou de degrés.
Entrer la valeur de déclenchement dans la zone de saisie, puis utiliser le menu déroulant pour indiquer si la région « vraie » doit être supérieure ou inférieure à la valeur de déclenchement.
La zone jaune de rétroaction indique le nombre actuel de rotations entières ou de degrés.
Cliquer sur le bouton Réinitialiser pour remettre le compteur à zéro.
La valeur de rétroaction est affichée en texte noir pour les rotations ou les degrés dans la direction positive (vers l'avant), et en texte rouge dans la direction négative (vers l'arrière).

Le *point de déclenchement* est la valeur précise, dans une plage de nombres, où se produit un changement de conditions. Par exemple, vous pouvez programmer le robot pour qu'il s'arrête quand le nombre de rotations devient supérieur à 10. Le point de déclenchement vaut dans ce cas 10 rotations.

Vous pouvez définir le *point de déclenchement* en entrant un nombre dans la zone de saisie. Pour définir la plage (au-dessus ou en dessous du point de déclenchement) qui générera le signal « vrai », utilisez le menu déroulant pour choisir l'opérateur supérieur ou inférieur (au point de déclenchement).

Par défaut, le bloc Capteur de rotation est paramétré pour qu'un nombre de degrés supérieur à 360 génère un signal « vrai ».

La zone de rétroaction compte le nombre de degrés ou de rotations selon lequel votre moteur tourne. Par défaut, cette zone affiche les degrés. Si la propriété Durée est fixée à Rotation, le compteur affichera des rotations entières. Cliquez sur le bouton Réinitialiser pour le remettre à zéro. Pour recevoir une rétroaction, assurez-vous que le moteur est connecté au port approprié et que la communication a été établie avec le NXT.

Fig.13

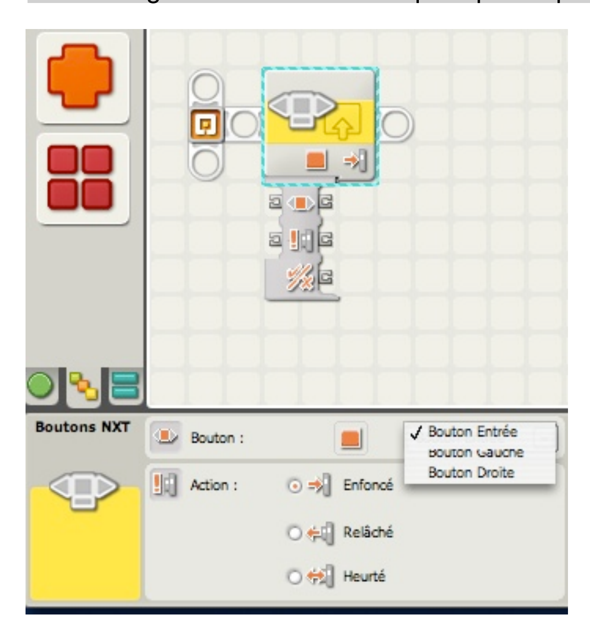
Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Capteur de rotation :

	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Port	Numérique	1 - 3	1 = A, 2 = B, 3 = C	
	Point de déclenchement	Numérique	0 - 2147483647	Valeur de référence de la comparaison	
	Direction du point de déclenchement	Logique	Vrai/Faux	Direction à utiliser pour la comparaison Vrai = Avant, Faux = Arrière	
	Supérieur/Inférieur	Logique	Vrai/Faux	Logique utilisée dans la comparaison : Vrai = Supérieur, Faux = Inférieur	
	Réinitialiser	Logique	Vrai/Faux	Vrai = Réinitialiser, Faux = Lire	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Direction	Logique	Vrai/Faux	Vrai = Avant, Faux = Arrière	
	Degrés	Numérique	0 - 2147483647	Valeur d'échelle lue depuis le capteur	

Bloc Boutons NXT

Ce bloc envoie un signal « vrai » via un fil de données lorsqu'un des boutons NXT est activé. Vous devez sélectionner le bouton et l'action qui enverront le signal « vrai ».

Fig.14



Zone Bouton: Indiquer quel bouton NXT envoie un signal « vrai » lorsqu'il est activé.
Zone Action: Choisir *Heurté* si vous voulez que le bouton NXT soit activé lorsqu'il est enfoncé puis relâché rapidement. Choisir *Enfoncé* si vous voulez que le bouton soit activé dès qu'il est enfoncé. Choisissez *Relâché* si vous voulez que le bouton soit activé dès qu'il est relâché.

3. La zone jaune de rétroaction affiche 1 lorsque le bouton NXT choisi est heurté, enfoncé ou relâché (en fonction de la configuration sélectionnée).

Fig.15

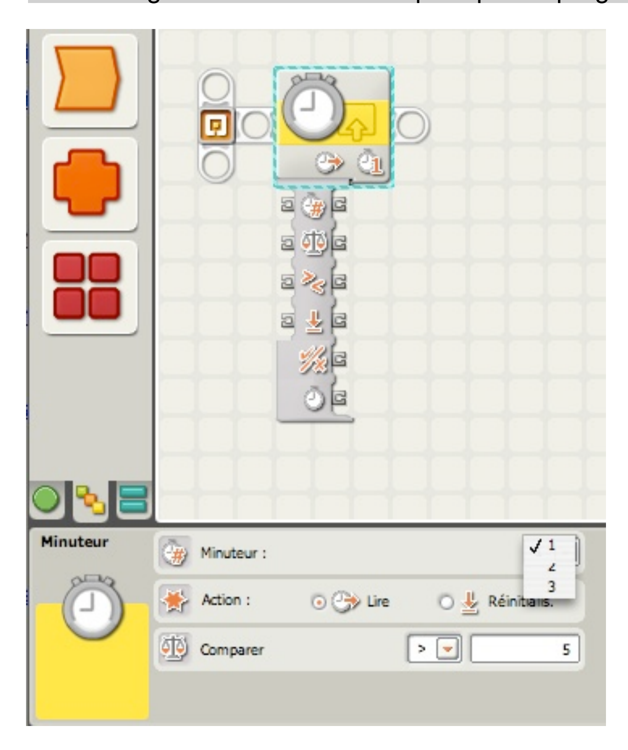
Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Boutons NXT :

	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Bouton	Numérique	1 - 3	1 = Droite 2 = Gauche 3 = Sélection	
	Action	Numérique	0 - 2	0 = Enfoncé 1 = Relâché 2 = Heurté	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	

Bloc Minuteur

Le NXT met à votre disposition 3 minuteurs (chronomètres) intégrés qui permettent de déclencher des actions en fonction du temps écoulé. Ces capteurs n'existent pas physiquement, mais ils sont activés grâce à un bloc de programmation.
Lorsque votre programme démarre, les trois minuteurs commencent automatiquement à compter. Ce bloc permet de choisir si vous voulez lire la valeur actuelle d'un minuteur ou forcer un minuteur à repartir de zéro.

Fig.16



Ce bloc peut, via des fils de données, envoyer la valeur actuelle du minuteur, ainsi qu'un signal logique (vrai/faux) qui varie selon que la valeur du minuteur est supérieure ou inférieure à un *point de déclenchement*.

Le *point de déclenchement* est la valeur précise, dans une plage de nombres, où se produit un changement de conditions. Par exemple, vous pouvez programmer votre robot pour qu'il s'arrête quand le minuteur dépasse 20 secondes. Le point de déclenchement vaut dans ce cas 20 secondes.

Vous pouvez définir le point de déclenchement en entrant un nombre dans la zone de saisie. Pour définir la plage (au-dessus ou en dessous du point de déclenchement) qui générera le signal « vrai », utilisez le menu déroulant pour choisir l'opérateur supérieur ou inférieur (au point de déclenchement).

Pour que le bloc Minuteur fonctionne, vous devez tirer un fil de données depuis le plot de données du bloc, puis le connecter au plot de données d'un autre bloc. Vous pouvez tirer deux types de fils de données en sortie depuis le plot de données du bloc Minuteur : un fil de données logiques, qui transmettra un signal vrai/faux, et un fil de données numériques (#) qui transmettra la valeur actuelle du minuteur.
Si par exemple, vous configurez le bloc Minuteur plus grand que 10 secondes pour le minuteur 1, le plot logique fournira une valeur Faux tant que les 10 secondes ne seront pas écoulées. Une fois cette valeur dépassée le plot logique fournira une valeur Vrai.

Vous pouvez également fournir la valeur de déclenchement de manière dynamique en connectant un fil de données en entrée au plot de données du bloc Minuteur.

Fig.17

Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Minuteur :

	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Minuteur	Numérique	1 - 3	1 = Minuteur 1 2 = Minuteur 2 3 = Minuteur 3	
	Point de déclenchement	Numérique	0 - 100	Valeur de référence de la comparaison	
	Supérieur/Inférieur	Logique	Vrai/Faux	Logique utilisée dans la comparaison : Vrai = Supérieur, Faux = Inférieur	
	Réinitialiser	Logique	Vrai/Faux	Vrai = Réinitialiser le minuteur Faux = Lire le minuteur	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Valeur du minuteur	Numérique	0 - 4294967296	Valeur du minuteur, en millisecondes	

Vous venez d'apprendre comment utiliser ces blocs au travers de leurs panneaux de configuration. Le vrai pouvoir de ces blocs se manifeste en les conjuguant avec d'autres blocs tels que **Boucle**, **Attendre** et **Commutateur**.
Nous verrons plus tard comment utiliser les **Boucles** pour répéter certaines tâches. Grâce à cet outil, vous accomplirez des répétitions à l'infini jusqu'au moment où vous déciderez d'arrêter le programme. Mais vous pourrez aussi configurer une boucle de telle sorte qu'elle s'interrompe quand certaine condition est rencontrée. Par exemple si le capteur photosensible détecte un niveau lumineux inférieur à 20, ou le capteur sonore détecte un bruit supérieur à 80, ou peut-être un moteur ayant subit une rotation supérieure à 20 degrés (capteur de rotation).
Il en sera de même avec les blocs **Attendre** et **Commutateur** qui donneront à vos robots les pouvoirs de décision.

Avant de nous quitter (provisoirement) retenez cette nouvelle règle:
6ème règle: Les blocs de programmation ne peuvent répondre qu'à une condition à la fois, en ne fournissant qu'un type de réponse logique: Vrai ou Faux.
Il faut autant de blocs que de conditions à satisfaire.



Leçon n°6: ATTENDRE et ATTENDRE…

Ce que fait votre robot le plus utilement, c'est *attendre…*
Oui, vous m'avez bien lu! Vous ne me croyez pas, alors voyons les choses différemment.

* BONG se dirige vers une ligne noire tracée au sol, *attendant* que le capteur détecte cette ligne.
* BONG se prépare à lancer une balle dans une cible, *attendant* que le capteur tactile soit appuyé puis relâché.
* BONG se dirige vers le mur, *attendant* que le capteur à ultrasons détecte ce mur.
* BONG est immobile sur la ligne de départ, *attendant* que le capteur sonore reçoive mon mot d'ordre "aller".

Avez-vous saisi *qu'attendre* est une partie importante de la programmation d'un robot? Que vos robots seront probablement en attente permanente d'un ou plusieurs événements? Cela peut-être l'attente d'un bouton du NXT à appuyer, ou quelque chose de comparable aux exemples précédents.
Toutes ces attentes seront traduites par l'utilisation du bloc **ATTENDRE** .

Bloc ATTENDRE

Un des principes les plus importants de ce bloc est qu'il *interrompra cette attente dès que la condition spécifique sera réalisée.*
Peu importe le capteur, capteurs physiques ou intégrés, ils sont tous concernés. Quand vous emploierez le bloc ‘Attendre’, vous devrez impérativement préciser quelle condition doit être réalisée pour mettre fin à cette attente.

Essayons de comprendre ce fonctionnement en adressant à BONG quelques instructions en pseudo code.
LEO > BONG avance jusqu'à ce que quelque chose survienne.

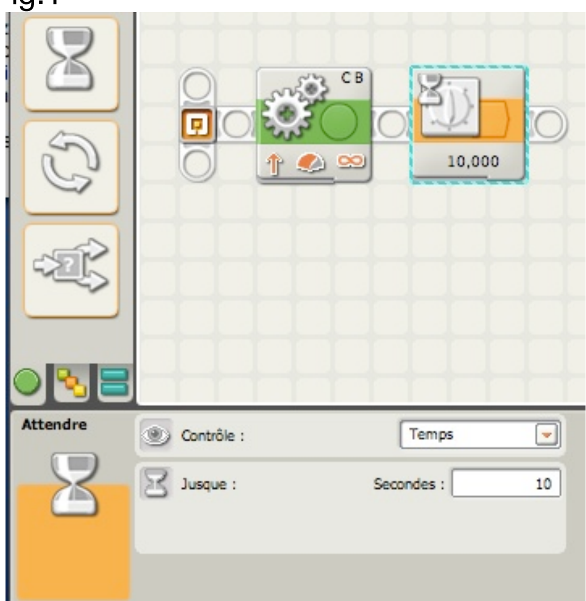
C'est vague, non? Que signifie "quelque chose survienne"? En fait, cela peut-être n'importe quoi:

- après un délai de 5 secondes,
- Dès que le capteur sonore enregistre un bruit intense,
- Dès que le capteur à ultrasons détecte un obstacle à 20 cm au devant.

Avez-vous d'autres idées?
Je souhaite que BONG avance jusqu'à ce qu'une condition particulière soit appliquée. Et avec NXT-G, cette condition peut-être satisfaite grâce aux capteurs, à un minuteur, aux boutons du NXT ou à un limiteur de temps.

Nous allons faire une petite démonstration:
Vous allez créer ce petit programme.

Fig.1



Le bloc DEPLACER est configuré pour une durée "*illimitée*" ce qui signifie qu'il fera tourner les moteurs B et C en permanence avec un déplacement vers l'avant.

Le bloc ATTENDRE est choisi dans la palette "commun"; il est aussi accessible dans la palette complète. En survolant l'icône, on affiche une petite palette dont on aperçoit le contenu dans la figure ci-dessous. Ce bloc offre plusieurs options correspondant aux différents capteurs connus et au capteur Temps. Tous ces choix sont associés à cette notion d'attente.

Fig.2



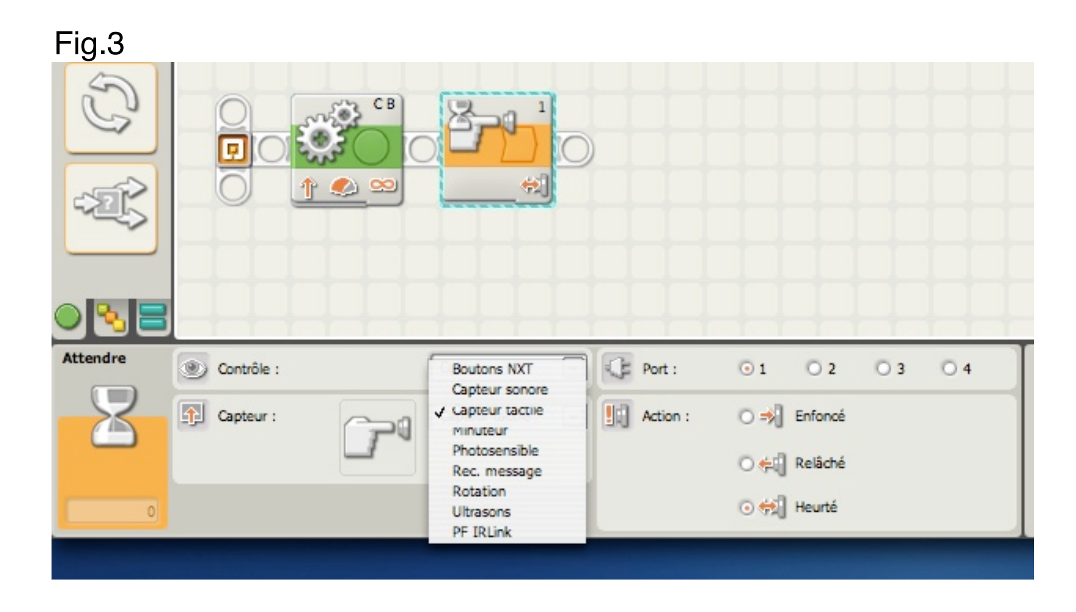
Le capteur TEMPS a été choisi pour cet exemple. Vous remarquerez dans le panneau de configuration que la *zone Contrôle* dispose justement un champ de saisie sélectionné sur TEMPS.
La *zone Jusque* a un champ de saisie dont la valeur a été fixée à 10 secondes.

Sauvegardez ce programme, téléchargez-le sur votre robot et exécutez-le. Que se passe-t-il?
Est-ce que le moteur tourne pendant 10 secondes, puis s'arrête? Si ce n'est le cas, vérifiez que le panneau de configuration du bloc Attendre est bien configuré sur 10 secondes.
La 4 ème règle s'applique ici. Malgré la rotation *illimitée* du moteur, le programme se termine après le dernier bloc.
Bien, c'est à peu près tout ce qu'il y a à dire sur le capteur Attendre TEMPS.

Voyons maintenant comment réagissent les capteurs physiques.
Remplacez le bloc Attendre Temps par le bloc Attendre Capteur tactile, puis exercez-vous à sauvegarder et utiliser ce programme modifié. Vous remarquerez que les panneaux de configuration changent selon le type de capteur.

Une petite astuce pour remplacer un capteur par un autre, sans procéder à une suppression, puis une nouvelle insertion.
Il suffit de sélectionner le bloc en place pour faire apparaître le panneau de configuration; puis, dérouler dans la *zone Contrôle*, le champ de saisie et choisir *Capteur* au lieu de *Temps*.
La *zone Jusque* est alors remplacée par la *zone Capteur*. Dérouler le champ et sélectionner l'élément concerné.
Il ne reste plus qu'à paramétrer le panneau de configuration, selon les conditions liées à ce capteur.

Fig.3



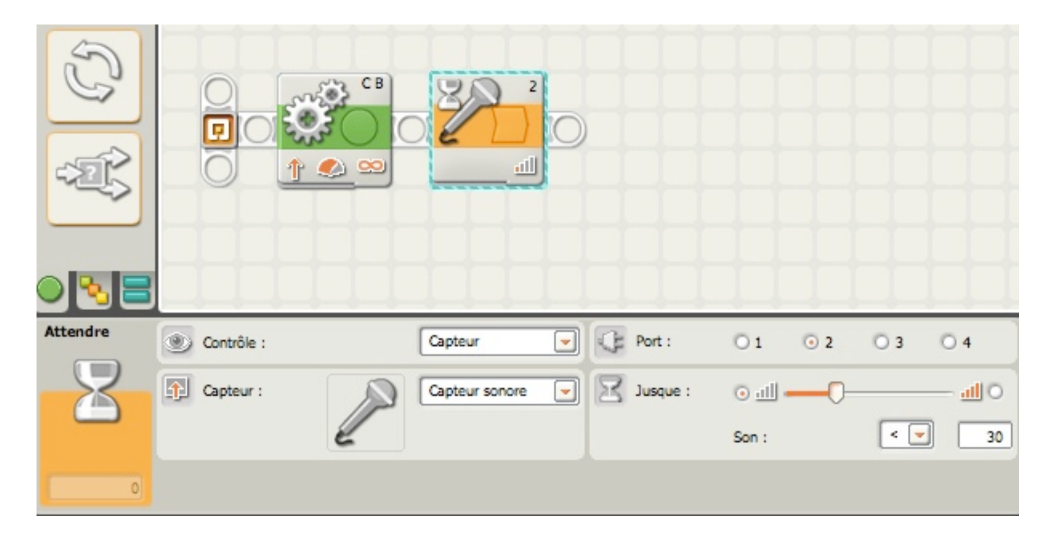
Nous allons maintenant examiner chacun de ces blocs liés aux capteurs sauf *Rec. message* qui est un bloc spécifique lié au dispositif Bluetooth, et que nous étudierons plus tard.

Bloc Attendre Capteur Tactile

Il est représenté ci-dessus avec la configuration suivante: câblé sur le port 1; fin d'attente pour Action Heurté. Dès que le bouton du capteur est heurté, les moteurs B et C s'arrêtent et BONG n'avance plus.

Fig.4

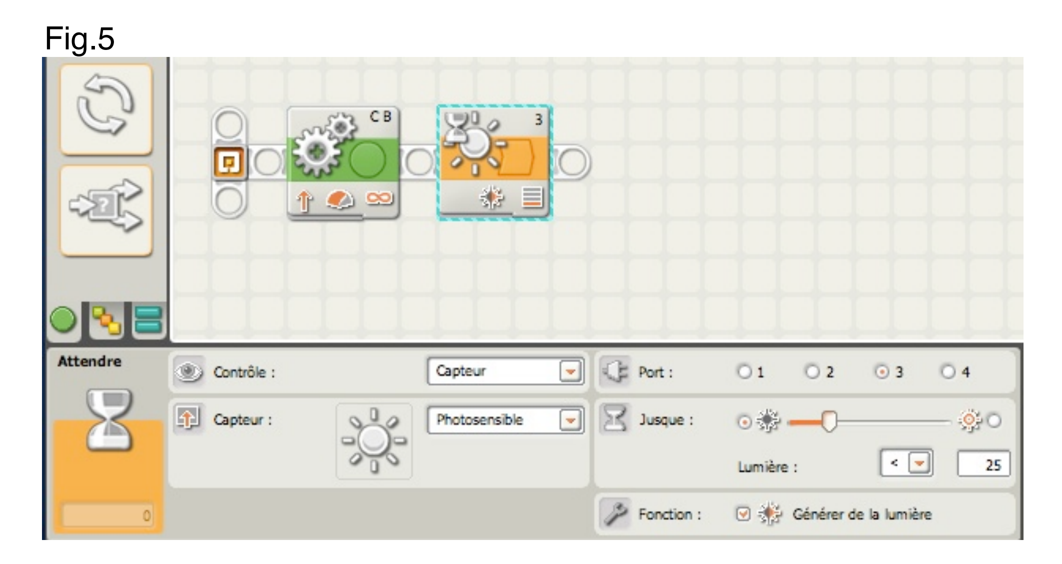
Bloc Attendre Capteur Sonore



Il est représenté ci-dessus avec la configuration suivante: câblé sur le port 2; fin d'attente pour Son < (inférieur) à 30. Tant que le niveau sonore est inférieur à 30, BONG avance. Dès que cette valeur est atteinte, les moteurs B et C s'arrêtent et BONG n'avance plus.

Bloc Attendre Capteur Photosensible

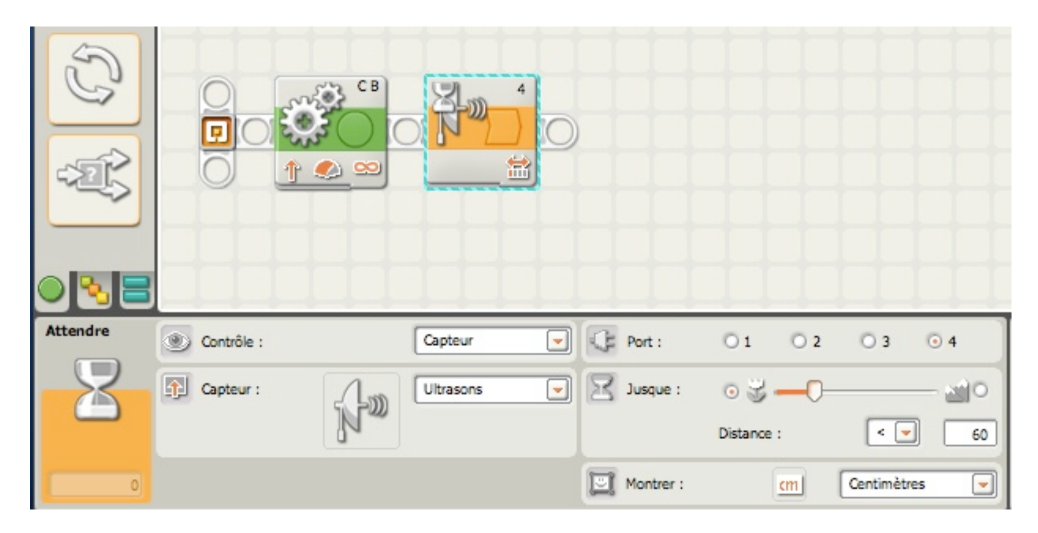
Fig.5



Il est représenté ci-dessus avec la configuration suivante: câblé sur le port 3; fin d'attente pour Lumière < (inférieur) à 25. Tant que le niveau lumineux est inférieur à 25, BONG avance. Dès que cette valeur est atteinte, les moteurs B et C s'arrêtent et BONG n'avance plus. A noter que le générateur de lumière est activé.

Bloc Attendre Capteur Ultrasons

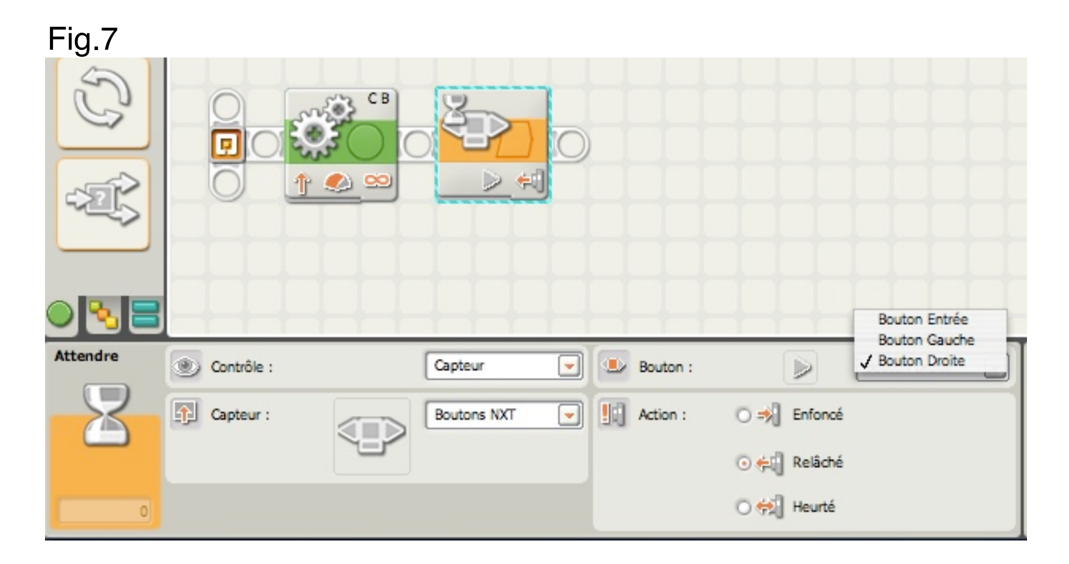
Fig.6



Il est représenté ci-dessus avec la configuration suivante: câblé sur le port 4; fin d'attente pour Distance < (inférieur) à 60 cm. Tant que cette distance est inférieure à 60 cm, BONG avance. Dès que cette valeur est atteinte, les moteurs B et C s'arrêtent et BONG n'avance plus.

Bloc Attendre Boutons NXT

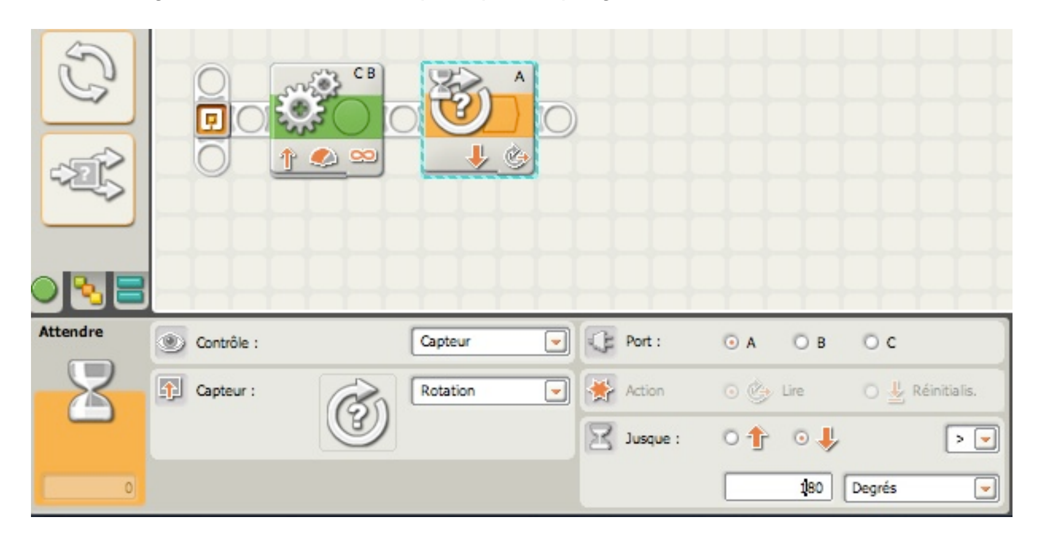
Fig.7



Il est représenté ci-dessus avec la configuration suivante: Attendre jusqu'à ce que le bouton flèche droite du NXT soit relâché. Quand cet événement survient, les moteurs B et C s'arrêtent et BONG n'avance plus.

Bloc Attendre Capteur Rotation

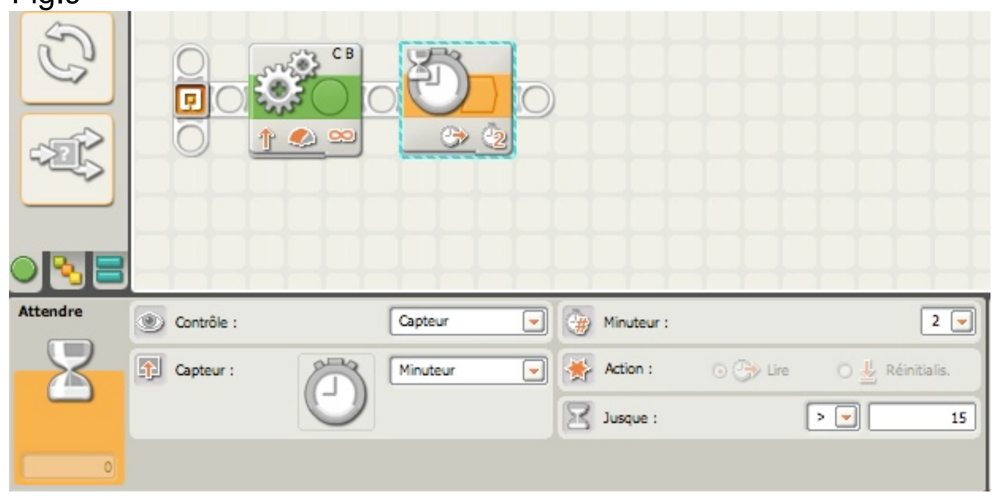
Fig.8



Il est représenté ci-dessus avec la configuration suivante: Attendre que le Moteur A accomplisse une demie rotation(180 degrés) en marche arrière. Quand cette action est accomplie, le moteur A s'arrête.

Bloc Attendre Capteur Minuteur

Fig.9



Il est représenté ci-dessus avec la configuration suivante: attendre jusqu'à ce que le Minuteur 2 dépasse 15 secondes.
Tous les Minuteurs se déclenchent dès le démarrage du programme. et le minuteur 2 attendra que 15 secondes soient écoulées avant de poursuivre le programme. Tant que cette valeur n'est pas atteinte, BONG avance. Dès que cette valeur est atteinte, les moteurs B et C s'arrêtent et BONG n'avance plus.
Vous apprendrez plus tard comment remettre à zéro les minuteurs, sachez pour l'instant qu'il y a 3 Minuteurs (Minuteur 1, Minuteur 2 et Minuteur 3) et que tous les trois commencent à compter dès le démarrage du programme.

Et maintenant, laissez-moi vous poser une question. Vous savez comment faire attendre votre robot, mais savez-vous comment lui faire exécuter et répéter encore et toujours d'autres actions?
Pour faire cela vous emploierez un nouveau bloc: le bloc BOUCLE. C'est ce que nous verrons dans une prochaine leçon.



Leçon n°7:
BOUCLER, ENCORE et ENCORE…

Nous sommes pratiquement arrivés à la fin de l'utilisation de la palette commune.
Nous allons maintenant aborder les blocs et les concepts qui permettent l'élaboration de programmes plus sophistiqués que ceux qui consistent à faire évoluer vos robots par de simples marches avant ou arrière, ou trajectoires circulaires.

Refaites-le à nouveau, …
Rejoignons à nouveau BONG. Supposons encore une fois qu'il dispose d'une paire d'oreilles et qu'il peut comprendre les ordres verbaux.
Je vais lui adresser une série d'instructions inhabituelles:
LEO > BONG, avance de 3 rotations (du moteur), arrête-toi et tourne à droite.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

LEO > BONG, avance de 3 rotations (du moteur), arrête-toi et tourne à droite.
(BONG avance à nouveau de 3 rotations, s'arrête puis tourne à droite…)

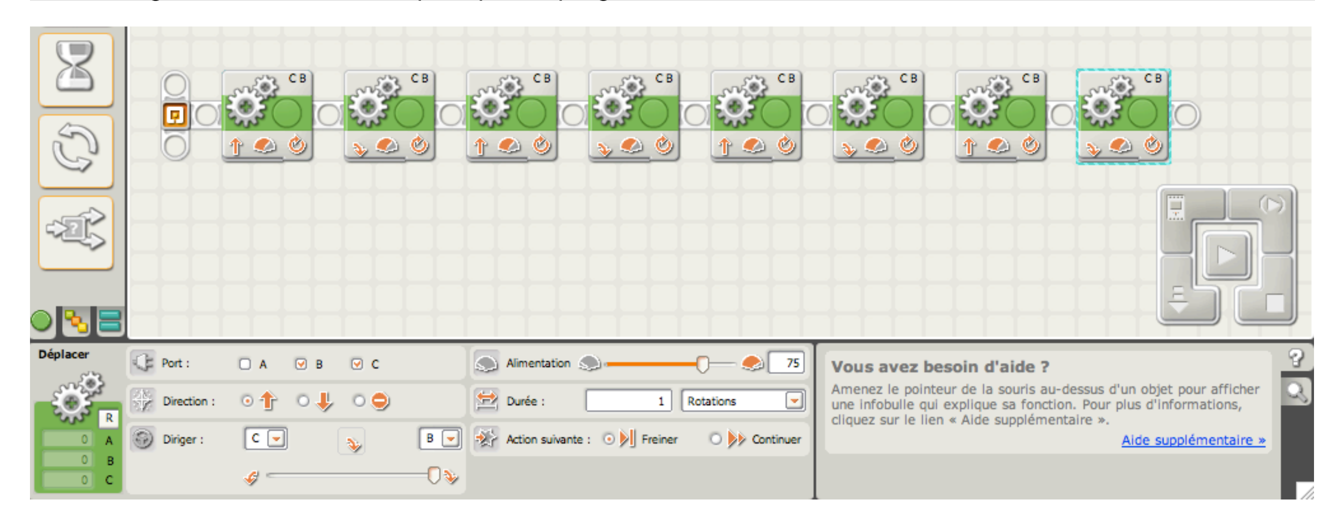
LEO > BONG, avance à nouveau de 3 rotations (du moteur), arrête-toi et tourne à droite.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

LEO > BONG, avance à nouveau de 3 rotations (du moteur), arrête-toi et tourne à droite.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

A votre avis, où se trouve maintenant BONG? C'est exact - , il est revenu à son point de départ.
Le chemin qu'il a suivi est celui d'un carré et il attend à nouveau mes instructions.

Si vous deviez écrire un programme NXT-G pour que BONG se déplace en suivant un tracé de forme carrée, vous auriez probablement placé 8 blocs "DEPLACER" sur le rayon de séquence, tels que présentés ci-dessous, et configuré chaque bloc DEPLACER avec les 2 mêmes propriétés. Exact?
Un bloc DEPLACER qui entraîne les moteurs B et C durant 3 rotations, et un 2ème bloc DEPLACER pour exécuter un virage à droite. Cette paire de blocs étant ensuite répétée 3 fois, soit un total de 8 blocs.

Fig.1



Bien, cela aurait sans doute marché. Mais c'est aussi dépenser beaucoup d'énergie pour que BONG suive un tracé en forme de carré pour revenir finalement à sa position de départ. Y aurait-il une meilleure solution?
Essayons d'améliorer le *pseudo-code*:
LEO > BONG, avance de 3 rotations (du moteur), arrête-toi et tourne à droite.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

LEO > BONG, répète ma première paire d'instructions 3 fois.

(BONG avance de 3 rotations, s'arrête puis tourne à droite…)
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

Beaucoup mieux! J'ai seulement lancé 2 instructions: une pour la marche, une autre pour la répétition (3 fois), et j'ai économisé ma voix! Cela n'a pas empêché BONG d'exécuter parfaitement le travail demandé.
Dans le *pseudo-code* j'ai donné à BONG un ordre et mis une *condition*.
Cette condition est tout simplement une *règle* que doit suivre BONG. Dans notre cas cette règle consiste à répéter 3 fois ce couple d'instructions. Tant que BONG a connaissance de la règle et qu'il la suive, peu importe le nombre de répétitions; cela peut-être 1 ou 100.

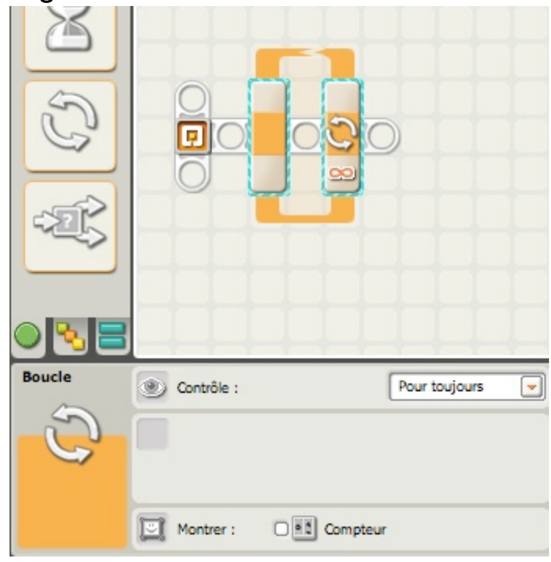
Au lieu de lui demander de répéter 3 fois la règle, puis-je utiliser une condition différente? Oui, évidemment.
Voici quelques exemples:
LEO > BONG répète ma première paire d'instructions jusqu'à ce que le capteur tactile soit heurté.
LEO > BONG répète ma première paire d'instructions jusqu'à ce que le Minuteur dépasse 45 secondes.
LEO > BONG répète ma première paire d'instructions indéfiniment.
C'est une autre façon de dire à BONG d'exécuter certaines instructions, et lui demander de les reproduire jusqu'à ce que "quelque chose survienne" (cela me rappelle la leçon n°6 …).
Et si BONG reproduit encore et encore ces instructions, on peut penser qu'il tourne en rond sur lui-même, qu'il boucle… C'est exactement ce qu'il fait: il BOUCLE sur lui même.
Et dans la vie de tous les jours, qu'est-ce une boucle? Ce peut-être un chemin circulaire, un rebondissement, un geste répétitif. De plus, qu'importe la dimension.

Si vous suivez un chemin circulaire, vous vous retrouverez forcément à votre point de départ. Une boucle de programmation est tout à fait semblable: elle tourne sur elle même, parfois indéfiniment jusqu'à ce que vous programmiez une *sortie* de boucle.

Bloc BOUCLE

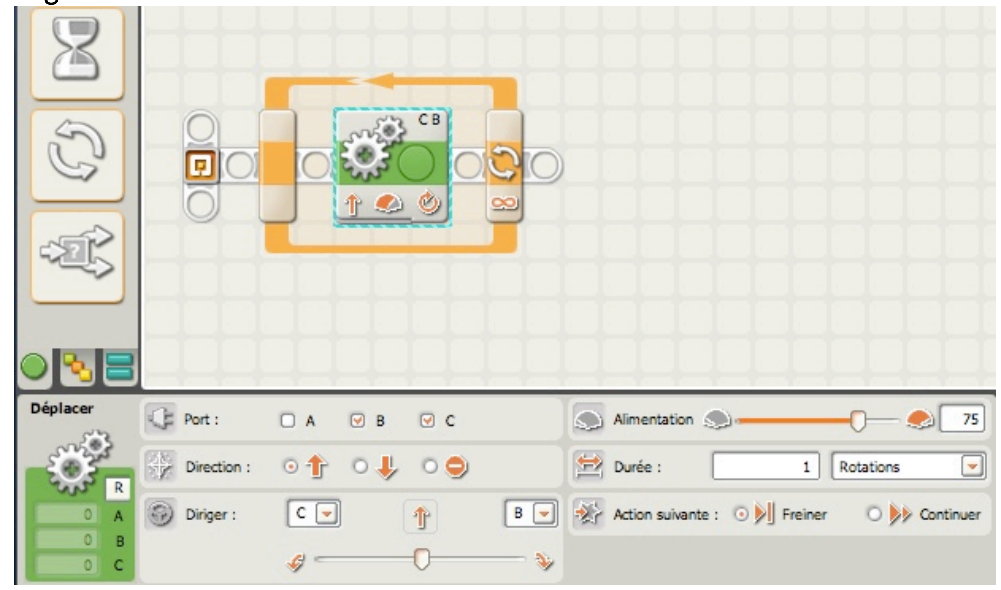
Et il se trouve que le logiciel NXT-G dispose d'un bloc spécialement conçu pour l'occasion: le bloc BOUCLE.

Fig.2



Pas très séduisant, ce bloc! Il est configuré dans la zone *Contrôle* sur *Pour toujours*.
Bien, maintenant chaque fois que j'introduirai à l'intérieur de cette boucle un bloc quelconque, ce dernier sera répété en permanence.
Prenons un exemple:
Je vais glisser à l'intérieur du bloc BOUCLE, un bloc DEPLACER ayant les caractéristiques suivantes: moteurs raccordés aux ports B et C, avec une rotation moteurs en marche avant.

Fig.3

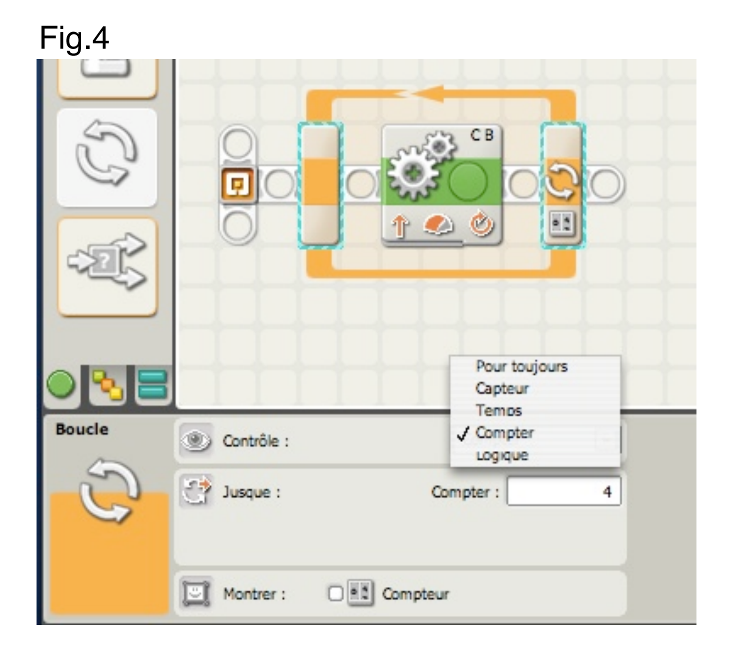


Vous avez déjà remarqué que le bloc BOUCLE s'adapte en s'élargissant pour recevoir le bloc DEPLACER. Il en sera toujours de même chaque fois qu'un nouveau bloc sera installé à l'intérieur de la BOUCLE.
Sauvegardons ce petit programme et faisons le exécuter par BONG.
BONG avance d'une rotation (de moteur), puis une très courte pause. Ensuite BONG avance encore d'une rotation, puis une nouvelle courte pause. Cela se poursuit de la même façon, jusqu'à ce que fatigué d'attendre ces répétitions, je décide d'arrêter le programme.
La pause survient quand le bloc DEPLACER termine son action et que le bloc BOUCLE vérifie sa condition. (cela se produit très rapidement).
N'oubliez pas que la condition est une règle que le robot doit suivre.
Pour ce bloc BOUCLE, la règle est ici de continuer "pour toujours" . Cela explique pourquoi le moteur tourne et tourne sans arrêt avec la courte pause à chaque tour.

Nous allons maintenant changer la condition. Comment dire à BONG de faire tourner le bloc DEPLACER 4 fois?
Jetez un coup d'oeil sur le panneau de configuration (ci-dessous). Vous remarquerez que la zone Contrôle dispose dans le champ de saisie d'un menu déroulant avec 5 options.

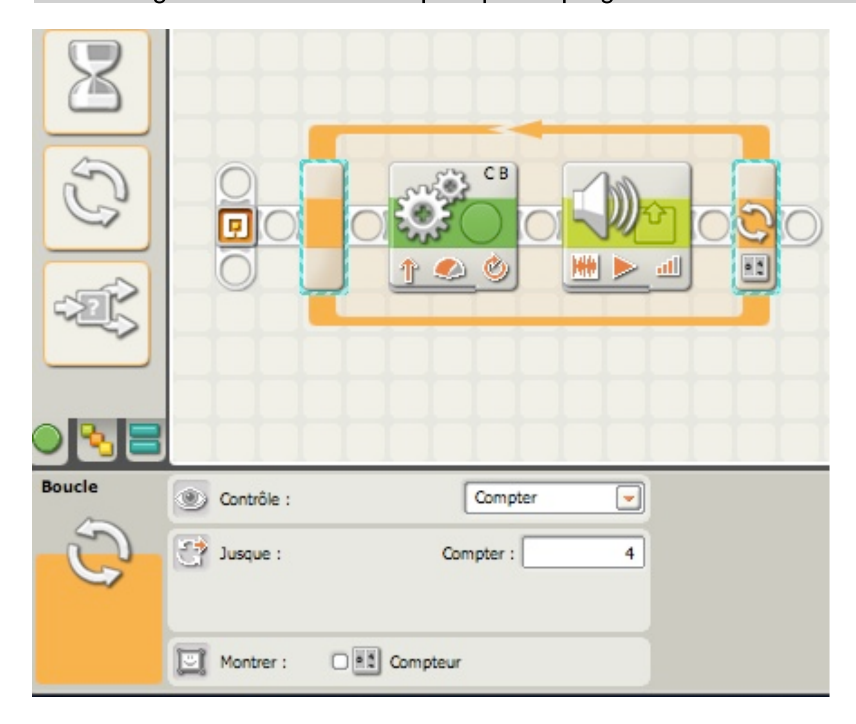
Option "Compter"

Fig.4



Choisissez "Compter", puis dans la zone de saisie "Compter", tapez 4. Vous devez taper ici une *valeur entière*. Les valeurs négatives ou zéro sont exclues.
Pour vérifier que le compte est bon, nous modifierons le programme en lui ajoutant un top sonore, qu'il suffira de compter pour la vérification.

Fig.5

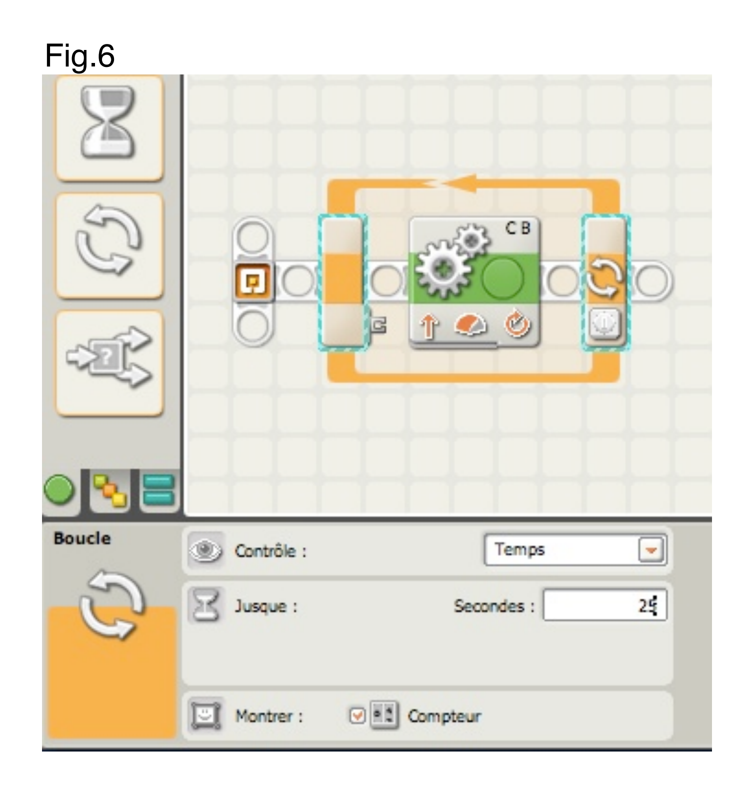


BONG va donc exécuter 4 rotations suivies chacune d'un top sonore.

Option "Temps"

Revenons à présent sur les 5 options proposées, et commençons par la plus simple "Temps".

Fig.6



Dans cette option, vous préciserez dans le champ "Secondes" le temps alloué à la boucle; ici 25 sec. Cela signifie que tous les blocs situés à l'intérieur de la BOUCLE agiront pendant ces 25 secondes. En fixant cette limite de temps, vous avez aussi signifié à la

BOUCLE sa *sortie* (loop break, en anglais). Au bout de 25 secondes, le programme se poursuivra ou s'arrêtera si aucun autre bloc ne suit le bloc BOUCLE.
Facile, non? La BOUCLE peut s'interrompre si vous annulez le programme, mais elle peut aussi être configurée pour s'interrompre si elle rencontre une condition particulière. C'est le cas déjà cité par exemple, d'un capteur tactile heurté.

Une remarque importante: s'il existe une ou des boucles à l'intérieur d'une boucle principale (on parle ici de "boucles imbriquées"), *toutes* les boucles internes seront exécutées avant que l'on sorte de la boucle principale.
Si par exemple, une boucle interne a une durée fixée à 40 secondes, elle sera exécutée, et la boucle externe (fixée à 25 sec.) ne sera pas répétée.

Autre remarque: dans la zone "Montrer", une case à cocher se trouve près de la petite icône "Compteur". Si vous cochez cette case un plot de données apparaît sur la partie gauche du bloc. Vous avez activé le plot compteur qui fournit à chaque reprise de la boucle le nombre de répétition. Les valeurs correspondantes sont transmises par le plot, grâce à un fil de données, à d'autres blocs pour les besoins du programme.

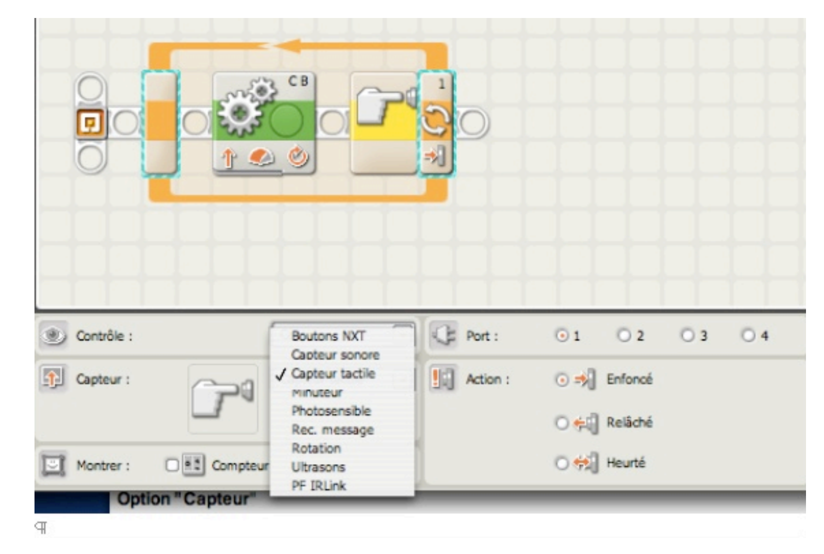
Option "Pour toujours"

Cette option a déjà été vue en début de la leçon, et elle est aussi simple que la précédente.
La boucle ne fait que répéter indéfiniment les bloc qu'elle contient et ne s'arrêtera que si le programme est interrompu ou si les piles sont hors service.

Option "Capteur"

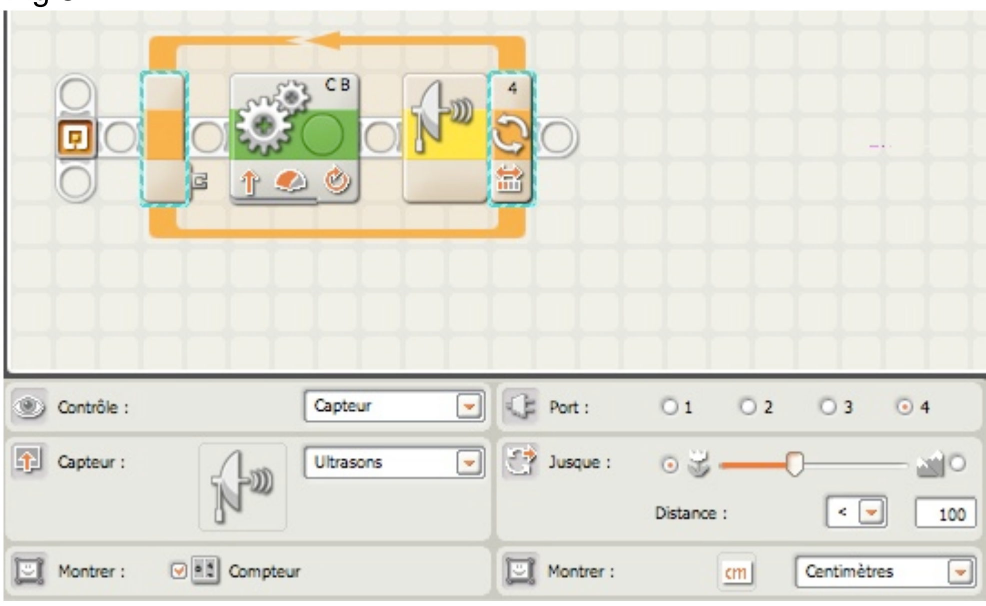
Vous allez à présent choisir dans la zone *Contrôle* l'option *capteur*.
Vous remarquerez que le *Capteur tactile* est sélectionné par défaut.
Mais dans la zone Capteur, le champ de saisie - grâce à un sous-menu déroulant - permet de sélectionner celui utile pour votre programme (On y trouve même ceux que vous avez acquis et installés en complément du kit # 8527 - PF IRLink dans notre cas en fin de liste).

Fig.7



Ils ont été déjà vus dans la leçon n°5 et sont d'un usage simple. La seule chose que vous devez déterminer est le *point de déclenchement* qui provoquera la sortie de boucle. Ceci a été également abordé dans cette leçon.
Nous allons prendre l'exemple du capteur à ultrasons.

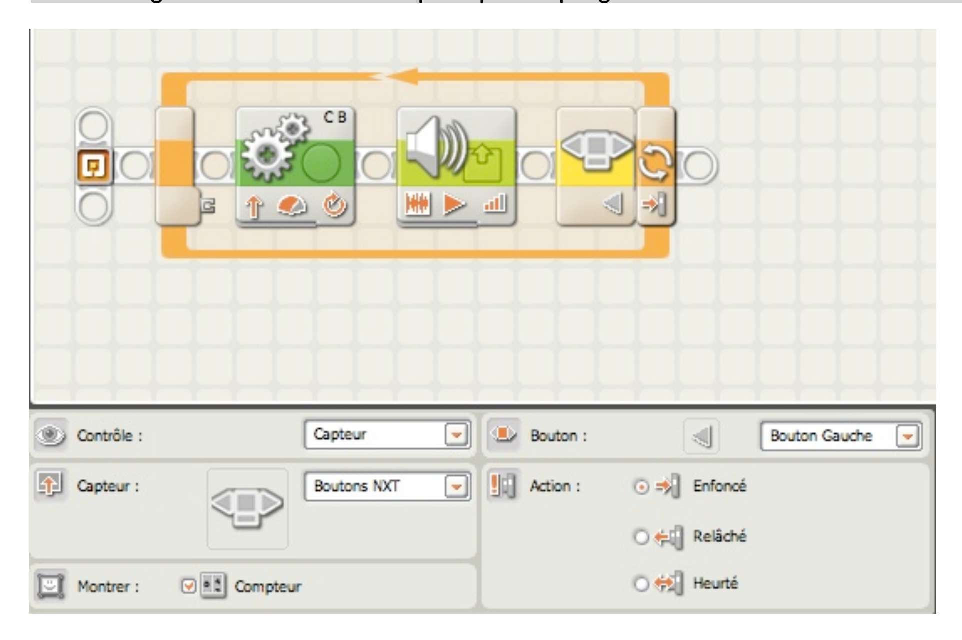
Fig.8



Dans cet exemple, le capteur à ultrasons est paramétré avec un point de déclenchement à 1,00 mètre. A noter également l'activation de la case compteur (le plot est visible à l'entrée de la boucle) pour permettre de récupérer la valeur du nombre de cycles.
Les bloc DEPLACER fait avancer le robot d'une rotation, suivi d'une légère pose, puis le cycle reprend tant que la distance entre le capteur fixé sur le robot, et un obstacle devant lui reste inférieure à 1 mètre. Dès que cette distance est atteinte, la boucle s'interrompt et le programme s'arrête.

Voici un autre exemple avec les boutons du NXT.
Le robot avance d'une rotation, émet un top sonore puis fait une légère pause. Cette action se répète inlassablement jusqu'au moment ou on appuie sur le bouton flèche gauche du NXT; la boucle s'interrompt alors et le programme se termine.

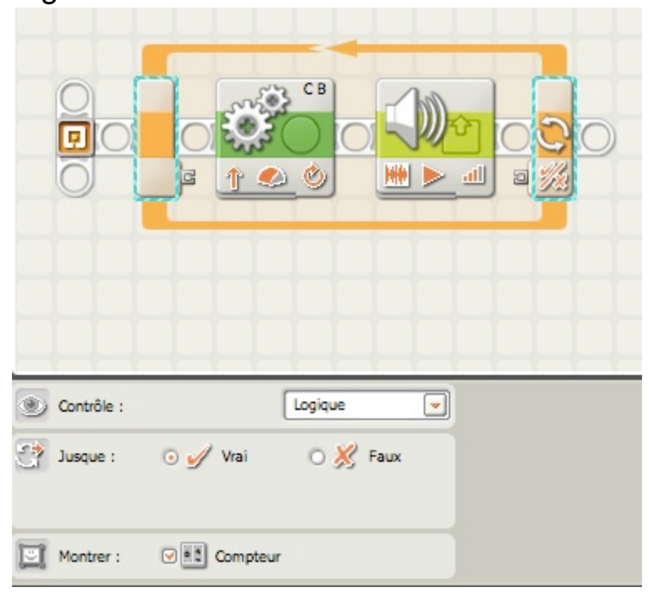
Fig.9



Option "Logique"

C'est a dernière option qui interrompt une boucle; elle se présente ainsi dans l'exemple qui suit:

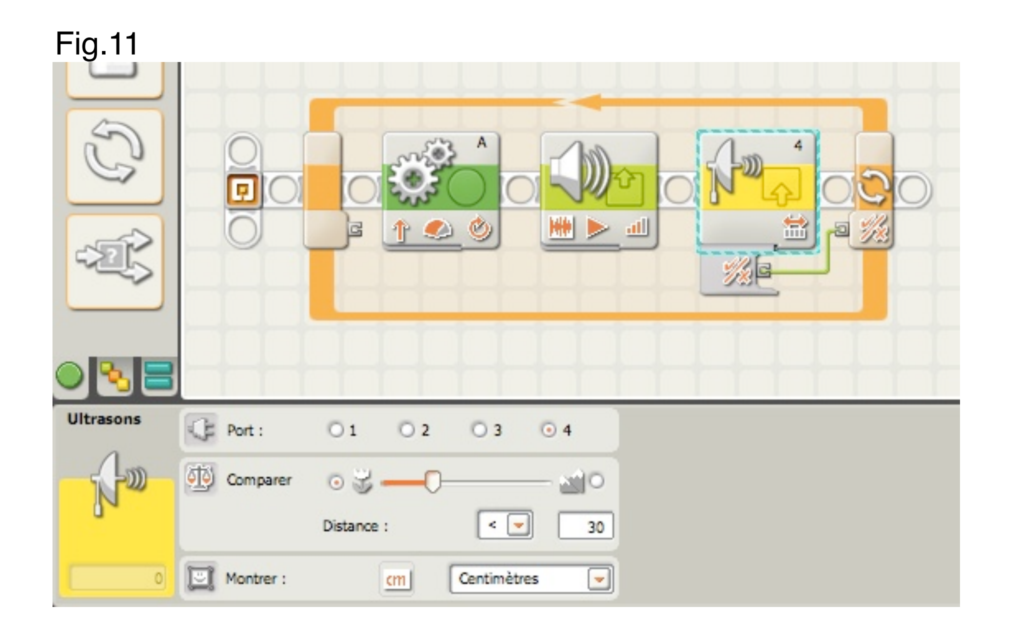
Fig.10



Plusieurs remarques:
D'abord, le plot de données à gauche de la boucle est visible, puisque la case *Compteur* a été cochée. Je rappelle que par ce plot on peut récupérer en *sortie*, une valeur numérique qui est le nombre de cycles de la boucle.
Ensuite, et c'est le seul cas, un plot de données à droite de la boucle, qui accepte *uniquement en entrée* une valeur logique OUI ou NON. La valeur de défaut est précisée dans la zone "Jusque" avec VRAI comme choix, mais on pourrait choisir aussi FAUX.
Cela signifie que la boucle s'exécute, quand la valeur sélectionnée (ici VRAI) est *reçue* (c'est une entrée de donnée).
En clair, tant que la valeur logique est VRAI, la boucle se répète. Dès que la valeur passe à FAUX, on sort de la boucle et le programme s'arrête.

Cet exemple illustre l'utilisation de cette option:

Fig.11



J'ai ajouté un bloc Ultrasons réglé suivant le panneau de configuration. Ensuite j'ai ouvert son Hub de plots de données pour relier le plot logique à celui de la boucle. J'ai enfin refermé le Hub, qui ne laisse visible que le plot utilisé, ici **v/x.**
Que se passe-t-il?
Le robot avance d'une rotation, émet un top sonore, puis fait une légère pause. Ensuite le capteur Ultrasons mesure la distance par rapport à un obstacle. Tant que la distance mesurée est inférieure à 30 cm, la boucle se répète. Dès que cette valeur est atteinte, on sort de la boucle et le programme s'arrête. Voilà une manière de sortir d'une boucle, sans intervention autre que celle du robot lui-même.
On aurait pu utiliser un autre capteur comme le Sonore qui aurait agit en fonction du niveau sonore (frapper des mains par exemple).

Je vous laisse maintenant écrire des programmes selon votre imagination et l'équipement de votre MOC. Exercez-vous pour maîtriser ce type d'action.

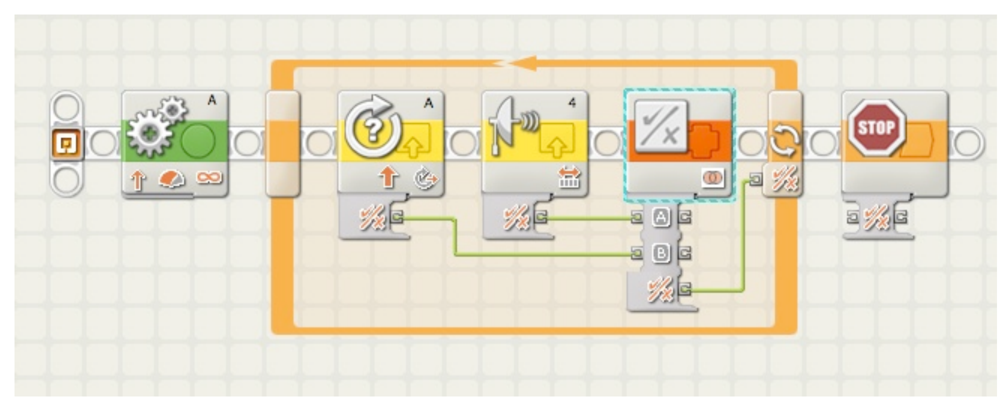
Nous allons à présent travailler sur quelques exemples.

Exemples

Exercice n°1:

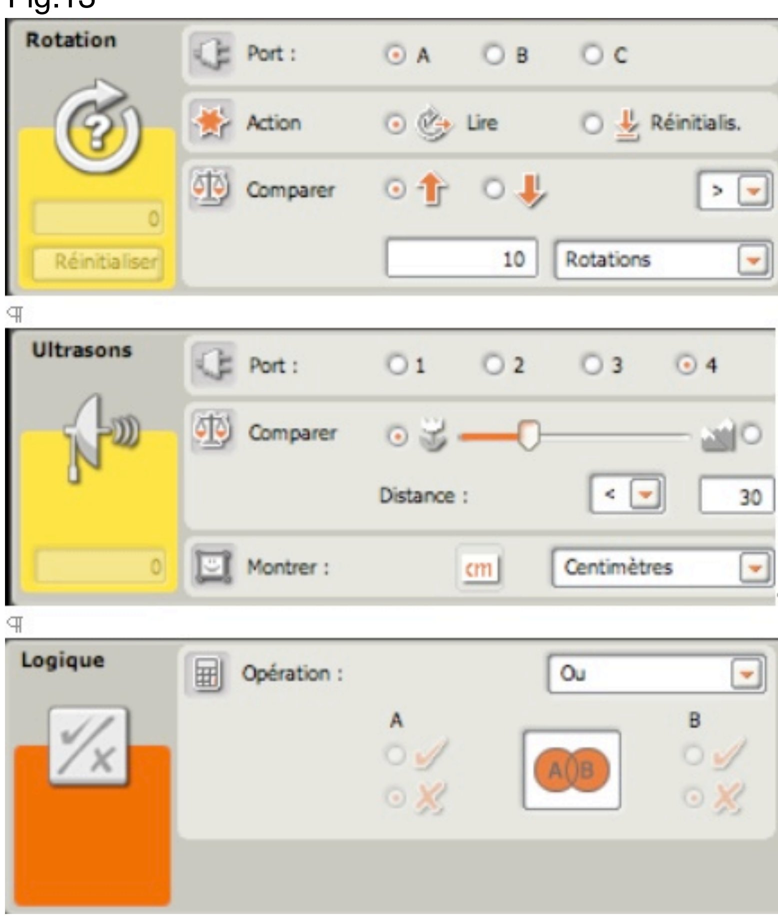
D'abord une réponse à une question posée par un utilisateur:
Rappel de son problème:
"… on voit que le robot avance de 10 rotations vers l'avant et dit qu'il voit quelque chose de trop près il recule de 5 rotations.
mais le problème c'est que la deuxième action ne se fait que si la première est finie, et je voudrais que l'action de reculer puisse se faire même pendant les 10 rotations."
Voici une proposition:

Fig.12



Les panneaux de configuration des blocs dans la boucle sont ainsi définis:

Fig.13



Le bloc LOGIQUE réunit les conditions de chacun des capteurs dans l'opération **OU** (qui s'utilise comme le bloc COMPARER). Cela permet justement la simultanéité des 2 conditions.
Je vous laisse le soin de tester ce programme.

Exercice n°2

Voici deux programmes strictement identiques sauf le fil de données qui existe dans le 2 ème cas.
(Le bloc STOP fait exactement ce qu'il est: il arrête le programme à l'endroit où il se trouve placé.)
A votre avis, comment se comporteront-ils?

Fig.14

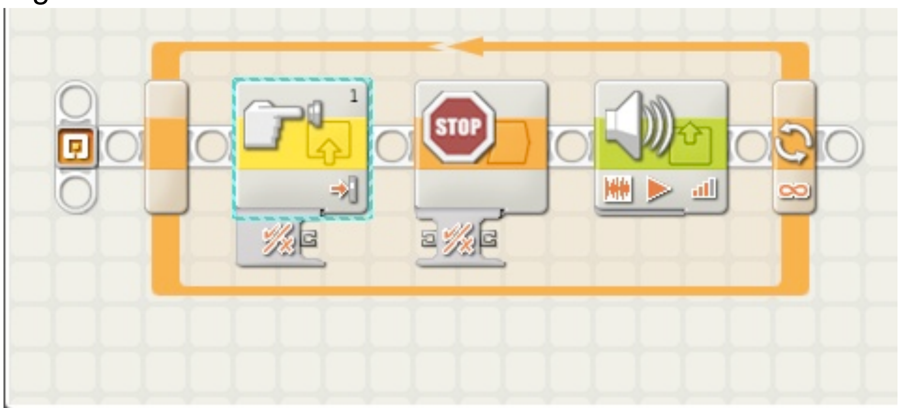
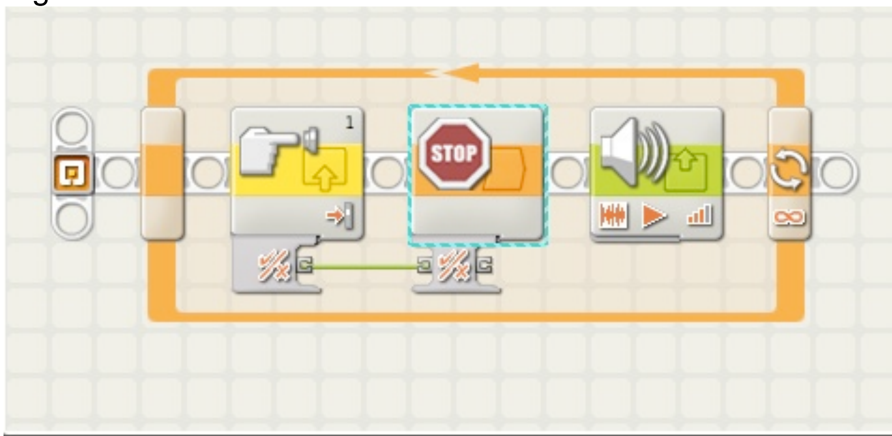


Fig.15

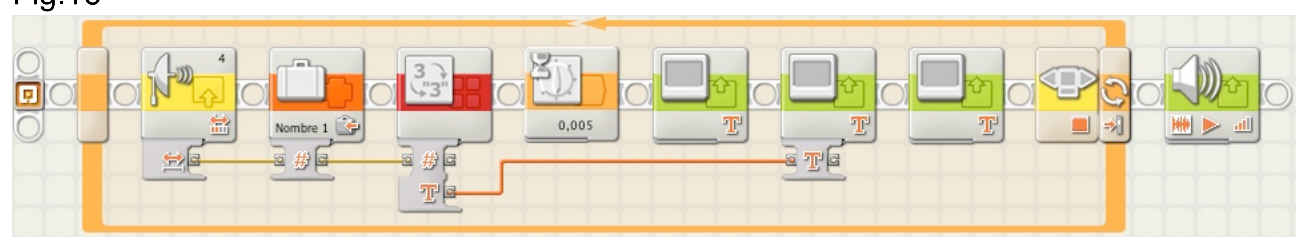


Dans quel cas le top sonore se fera-il entendre? Pourquoi?

Exercice n°3

Cette boucle affiche en permanence sur l'écran du NXT la distance mesurée par le capteur Ultrasons. La distance est stockée dans une variable (Nombre 1), puis elle est convertie en mode texte pour être affichée. Le bouton Entrée du NXT permet de sortir de la boucle. Cette sortie est confirmée par un top sonore.
Les deux blocs "AFFICHER" extrêmes comportent des textes fixes pour expliciter la valeur de la distance ("Distance Objet" et "cm")

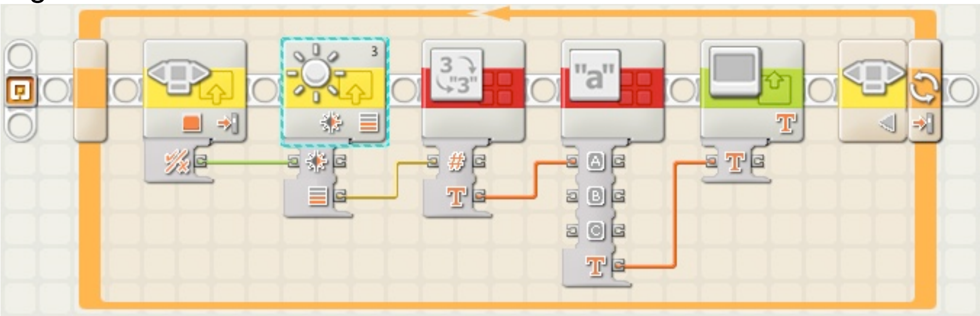
Fig.16



Exercice n°4

Cette boucle affiche la valeur du niveau lumineux en passant par le bloc TEXTE. Dans A la valeur de l'intensité en mode texte, en B un texte d'accompagnement (ici "Valeur Capt"), C vide. Le résultat apparaît sous la forme "65 Valeur Capt".
Une fois le programme lancé, la touche ENTREE permet de lire et d'afficher la valeur du capteur. Pour sortir de la boucle on appuie sur la flèche gauche. En réappuyant sur la touche ENTREE on relance la mesure. Par le jeu de ces 2 boutons, on arrête et relance le programme.

Fig.17



A vous maintenant d'imaginer une boucle faisant usage du bloc SON.

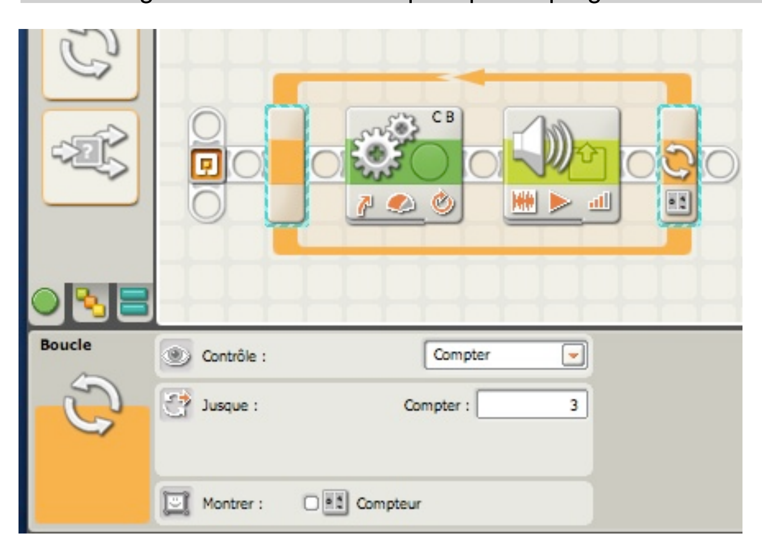
Boucles imbriquées:

Cette expression est souvent employée pour signifier que plusieurs boucles (chargées d'accomplir des actions différentes), sont astucieusement entremêlées pour relancer un processus complexe; le dispositif se répétant jusqu'à ce que vous décidiez d'y mettre fin. Pour expliquer ce concept, je vais imaginer un simple programme pour BONG dont j'aimerais bien vous confier la réalisation.

Mais d'abord, laissez-moi vous expliquer ce que BONG devrait faire en utilisant un *pseudo-code*:
LEO > BONG, je voudrais une avancée sur un tracé circulaire pendant 3,5 rotations du moteur, suivi d'un top sonore.
Recommencer cette manoeuvre encore 2 fois (soit 3 au total), puis, vérifier que le capteur photosensible détecte un niveau lumineux supérieur à 90.
Si le point de déclenchement n'est pas atteint, recommencer tout le processus.

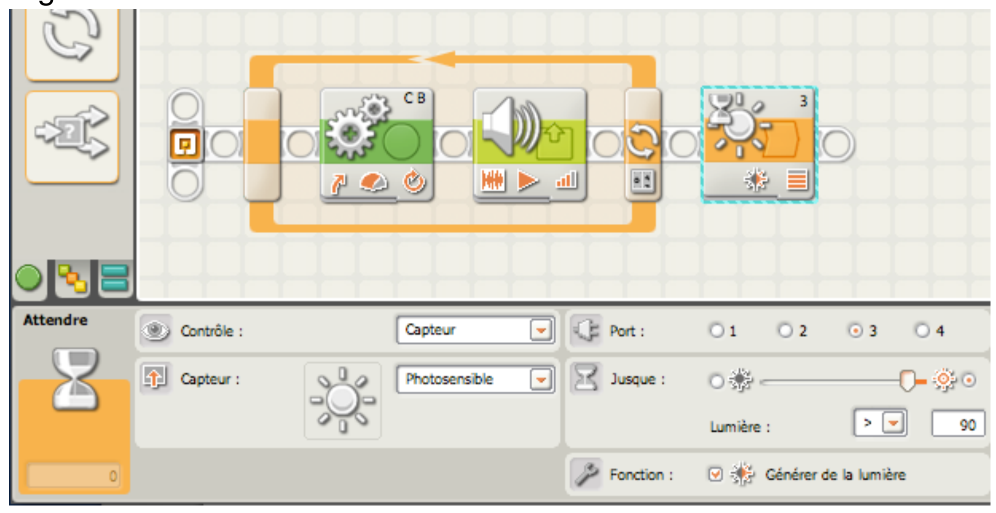
Je vais donc concevoir une BOUCLE comprenant un bloc DEPLACER et un bloc SON; cette boucle sera répétée 3 fois.

Fig.18



Je vais maintenant ajouter un bloc CAPTEUR PHOTOSENSIBLE et fixer le point de déclenchement à 90.

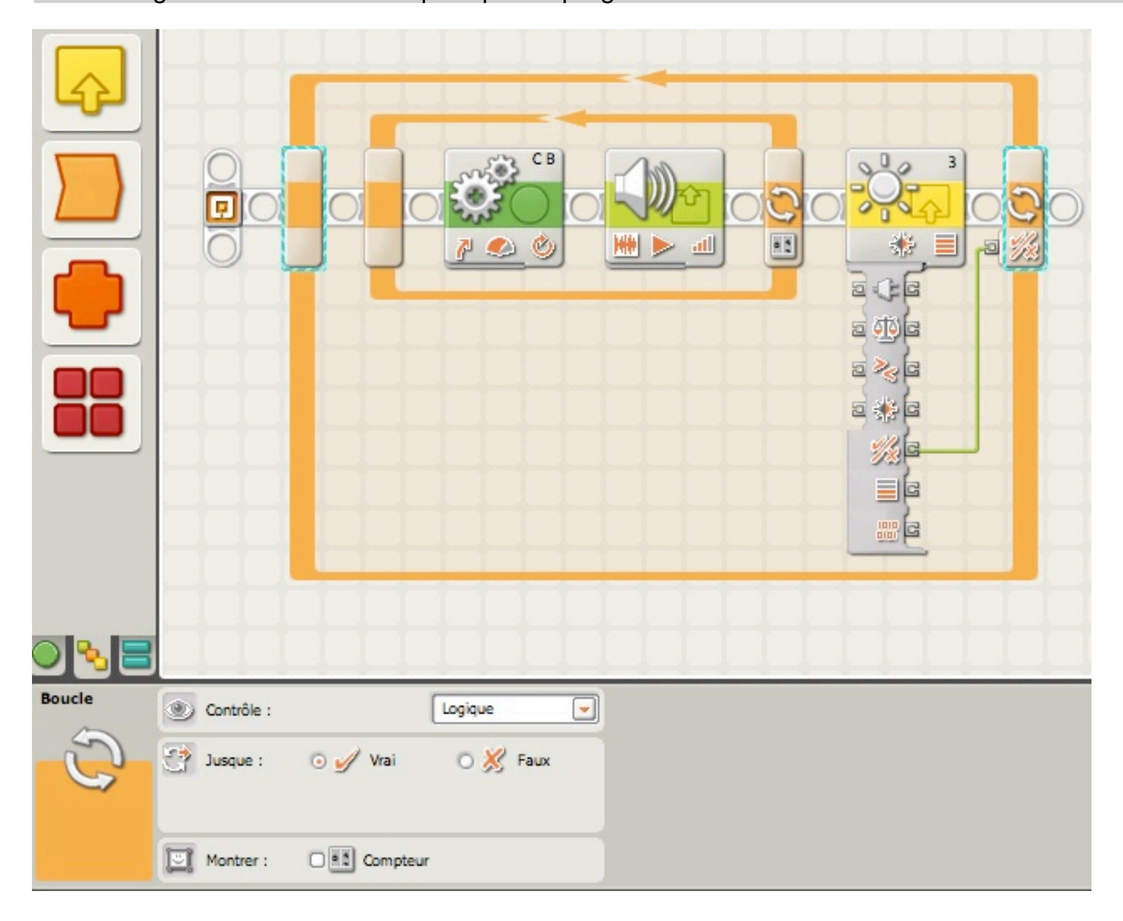
Fig.19



Mais maintenant, j'ai un problème. Si vous examinez le programme, on constate que la boucle se répète 3 fois avec à l'intérieur un bloc DEPLACER suivi d'un bloc SON, puis on sort de la boucle. Ensuite le bloc CAPTEUR PHOTOSENSIBLE est testé? Si le niveau lumineux est supérieur à 90, le programme se termine. Mais que se passe-t-il, si le niveau est inférieur à 90?
Comment puis-je obliger le programme à redémarrer?
Une des réponses, consiste à demander à BONG de recommencer le programme. Il faudra donc appuyer sur le bouton ENTREE du NXT pour faire redémarrer ce programme. Cela fonctionnera mais d'une façon laborieuse. Il doit bien y avoir une solution plus astucieuse.

Examinez à présent cette modification. Vous remarquerez que le petit programme se trouve INCLUS dans une autre BOUCLE. J'ai aussi remplacé le bloc CAPTEUR PHOTOSENSIBLE de la palette *Commun*, par le même capteur, mais pris cette fois-ci dans la palette *Complète* (pour disposer des plots de données).

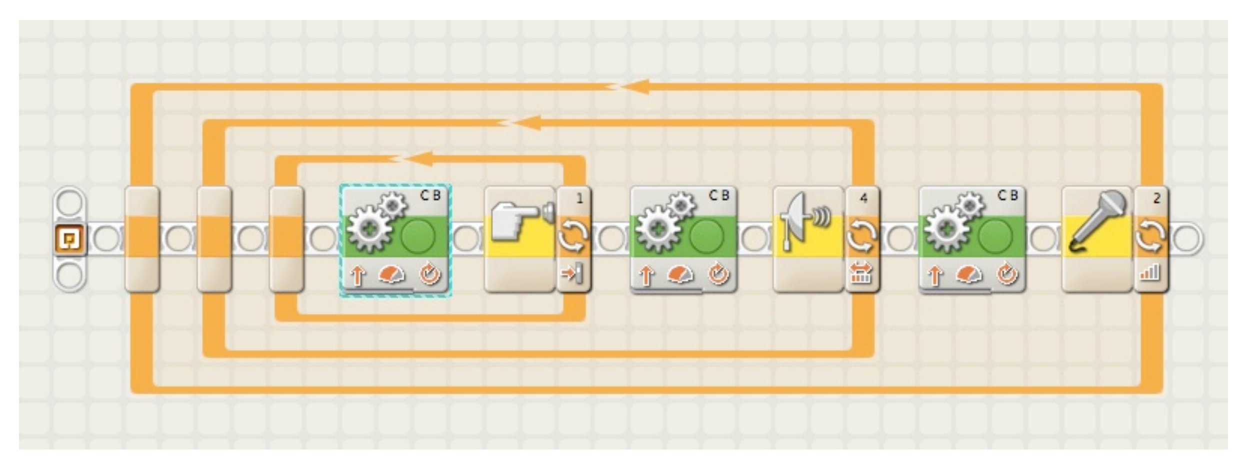
Fig.20



Ce que vous avez sous les yeux est une BOUCLE IMBRIQUEE, une BOUCLE dans une BOUCLE.
Quand le programme se déroule, chaque bloc à l'intérieur de la BOUCLE EXTERNE s'exécute. D'abord les blocs placés dans la boucle INTERIEURE répétés 3 fois (Blocs DEPLACER et SON). Puis on sort de cette boucle. Ensuite, le CAPTEUR PHOTOSENSIBLE envoie une information *logique* VRAI / FAUX vers le plot de données de la BOUCLE EXTERNE. Si le niveau lumineux détecté est inférieur à 90, la boucle n'est pas rompue et elle recommence son cycle. Dès que la valeur 90 est atteinte, on sort de la BOUCLE EXTERNE et le programme se termine.
Vous avez remarqué le fil de données reliant le plot **v/x** du bloc CAPTEUR PHOTOSENSIBLE à celui de la BOUCLE EXTERNE. Dans le panneau de configuration le contrôle se fait par une valeur Logique. Voici une méthode très utile pour sortir d'une boucle en utilisant un capteur; n'importe quel capteur.

Et pour terminer cette leçon, je vous propose cet exemple, une situation complexe. Pouvez-vous expliquer ce qu'il fait?

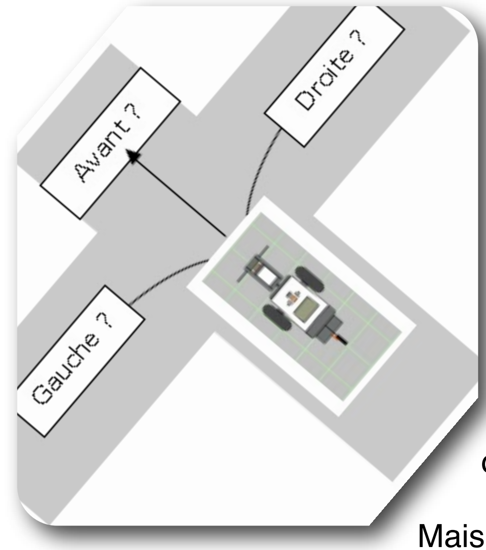
Fig.21



Oui, c'est une BOUCLE, dans une BOUCLE, dans une BOUCLE - 3 blocs BOUCLES IMBRIQUEES.
Quand on lance le programme, que constate-t-on?
D'abord débute la boucle du plus bas niveau: bloc DEPLACER jusqu'à ce que le capteur TACTILE soit actionné (ici Appuyé). La boucle est alors rompue.
Puis c'est la boucle intermédiaire qui se répète: un autre bloc DEPLACER jusqu'à ce que le point de déclenchement du capteur à ULTRASONS soit atteint. Arrivé à cette valeur la 2ème boucle est rompue à son tour.
Enfin tout se termine sur la boucle EXTERNE: un dernier bloc DEPLACER jusqu'à ce que le point de déclenchement du capteur SONORE est atteint. A ce moment, c'est la sortie de boucle et le programme s'arrête.

Rappelez-vous: dans le cas de BOUCLES IMBRIQUEES, le programme commence toujours par la boucle de plus bas niveau et se termine par la boucle de 1er niveau.

En conclusion, retenez que le bloc BOUCLE est un bloc puissant et très utile. Aussi, je vous conseille de vous exercer en utilisant toutes les options; vous l'emploierez souvent pour les nombreuses tâches répétitives de vos robots.





Leçon n°8: CHOISIR...DECIDER...

Cette leçon couvre un dernier bloc de la palette Commun. Lorsque vous l'aurez assimilée, vous serez en mesure d'offrir à vos robots des talents supplémentaires, notamment de faire des choix parmi de multiples et possibles actions.

Mais d'abord,

rappel des règles:

*** Première règle: familiarisez-vous avec les outils de programmation.**
*** 2ème règle: les capteurs doivent être connectés aux ports d'entrée (1,2,3 & 4) et les moteurs doivent être connectés aux ports de sortie (A, B & C).**
*** 3ème règle: Un bloc ne peut accomplir qu'une seule tâche à la fois. Il faut autant de blocs que de tâches à accomplir.**
*** 4ème règle: un programme NXT-G se termine une fois le dernier bloc exécuté.**
*** 5ème règle: les seules données qui peuvent circuler entre les blocs sont du TEXTE, des NOMBRES et du type LOGIQUE - et seulement ces trois.**
***6ème règle: Les blocs de programmation ne peuvent répondre qu'à une condition à la fois, en ne fournissant qu'un type de réponse logique: Vrai ou Faux.**
Il faut autant de blocs que de conditions à satisfaire.
*** 7ème règle: dans le cas de BOUCLES IMBRIQUEES, le programme commence toujours par la boucle de plus bas niveau (à l'intérieur) et se termine par la boucle de 1er niveau (à l'extérieur).**

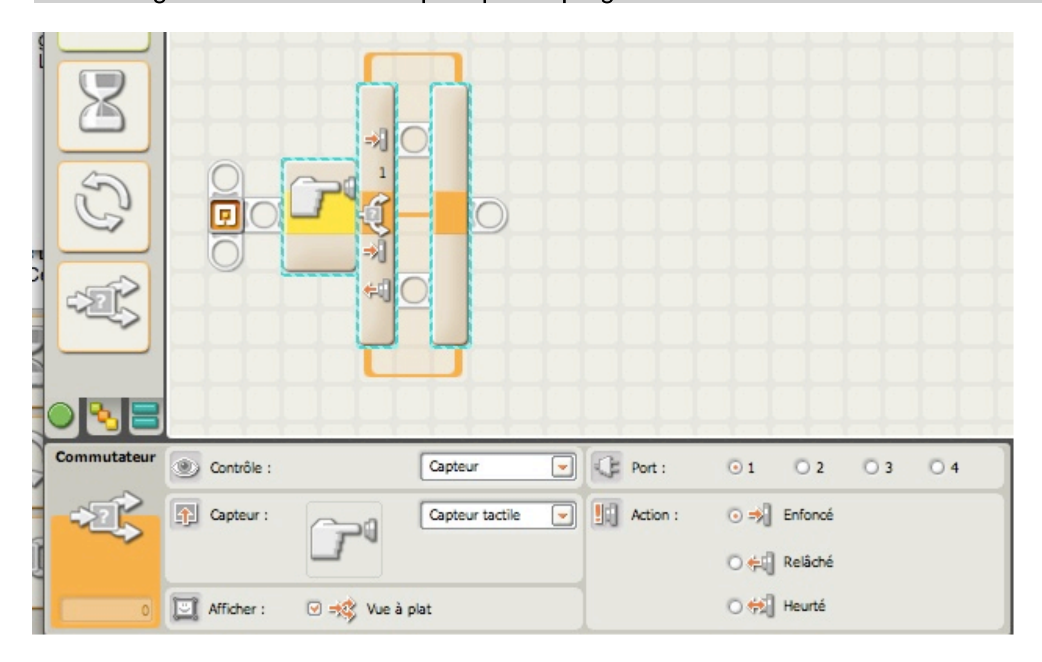
A Droite ou à Gauche? … Sortie A ou Sortie B? …

Pour commencer je vais adresser à BONG un *pseudo-code* pour cette nouvelle tâche.
LEO > BONG avance de 3 rotations (du moteur) puis stop. Si le capteur PHOTOSENSIBLE détecte un niveau lumineux supérieur à 30, tourne à droite. Sinon, tourne à gauche.
Vous savez comment programmer BONG pour le faire avancer à l'aide du bloc DEPLACER. Mais comment saisir la valeur issue du capteur PHOTOSENSIBLE et l'employer, pour aider BONG à prendre une décision: tourner à droite ou à gauche?
La réponse est simple: vous allez utiliser un nouveau bloc COMMUTATEUR.

bloc COMMUTATEUR

Voici à quoi il ressemble, présenté avec ses paramètres de défaut:

Fig.1



Ce bloc COMMUTATEUR utilisera une valeur en ENTREE pour déterminer son chemin. Cette valeur peut-être numérique, une portion de texte, ou une valeur Logique (VRAI ou FAUX).
Mais le plus intéressant est que le nombre de chemins n'est pas limité à 2. On pourrait par exemple configurer le bloc COMMUTATEUR pour répondre à ce *pseudo-code*:

LEO > BONG, prends une valeur au hasard comprise entre 1 et 5.

LEO > Si le nombre est 1, tourne à droite.

LEO > Si le nombre est 2, tourne à gauche.

LEO > Si le nombre est 3, fais un demi-tour.

LEO > Si le nombre est 4, fais un tour complet.

LEO > Si le nombre est 5, avance.

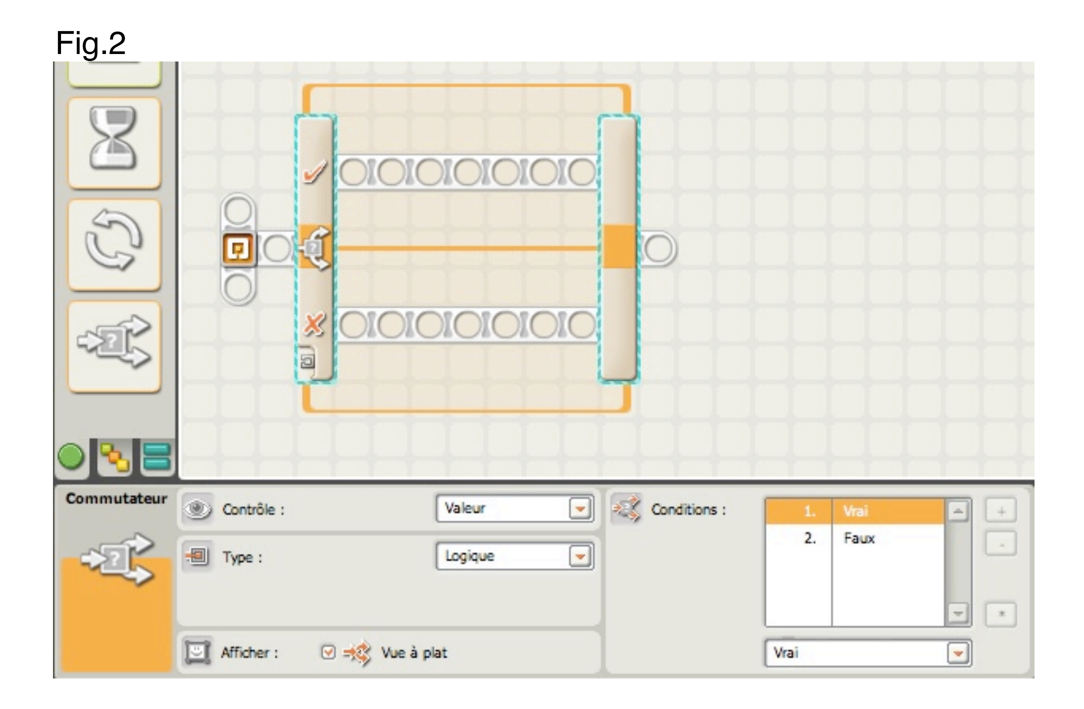
Dans cet exemple, j'ai demandé à BONG de choisir au hasard un nombre. Cela est possible en utilisant un bloc VARIABLE que nous avons déjà rencontré dans la leçon n°4 (nous en reparlerons d'une manière plus détaillée dans une autre leçon). Mais pour l'instant, supposons que BONG est capable de générer par lui-même ces nombres.
Bien, étant donné que BONG dispose de 5 nombres (1, 2, 3, 4 et 5), il existe 5 actions possibles qui leur sont associées. Je vais aussi utiliser le terme *chemin* au lieu de *action*, parce que le bloc COMMUTATEUR offre aux robots différents *chemins* possibles.

A chaque chemin suivi par le robot, correspond des actions différentes.
Un chemin peut ordonner au robot d'avancer, de vérifier, grâce au capteur à ULTRASONS, de la présence d'un obstacle.
Un autre chemin pourrait diriger le même robot vers une direction opposée, attendant que le capteur TACTILE soit heurté, tout en comptant le nombre de rotations du moteur.
C'est la grande affaire de ce bloc: chaque chemin dispose de blocs de programmation spécifiques qui donnent au robot encore plus de puissance. Vous pouvez même ajouter à

l'intérieur d'un chemin un nouveau bloc COMMUTATEUR, multipliant à l'infini les chemins…

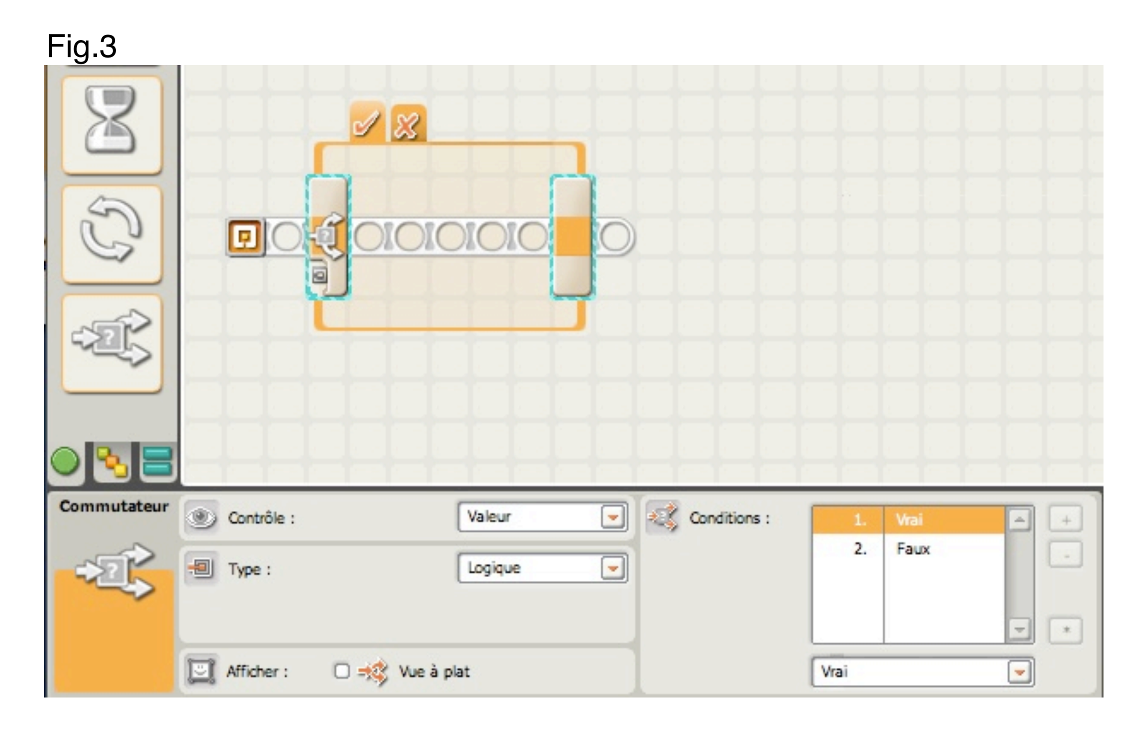
Avant de voir comment fonctionne ce bloc, je dois préciser un point particulier figurant dans le panneau de configuration:
pour les besoins d'explication, j'ai choisi dans la zone *Contrôle* du panneau de configuration *valeur,* type *Logique.*

Fig.2



Vous noterez dans la zone *Afficher* que la case *Vue à plat* est cochée.
Décochez cette case.
La présentation graphique du bloc change quelque peu et se présente ainsi:

Fig.3



La différence réside dans l'affichage des chemins.

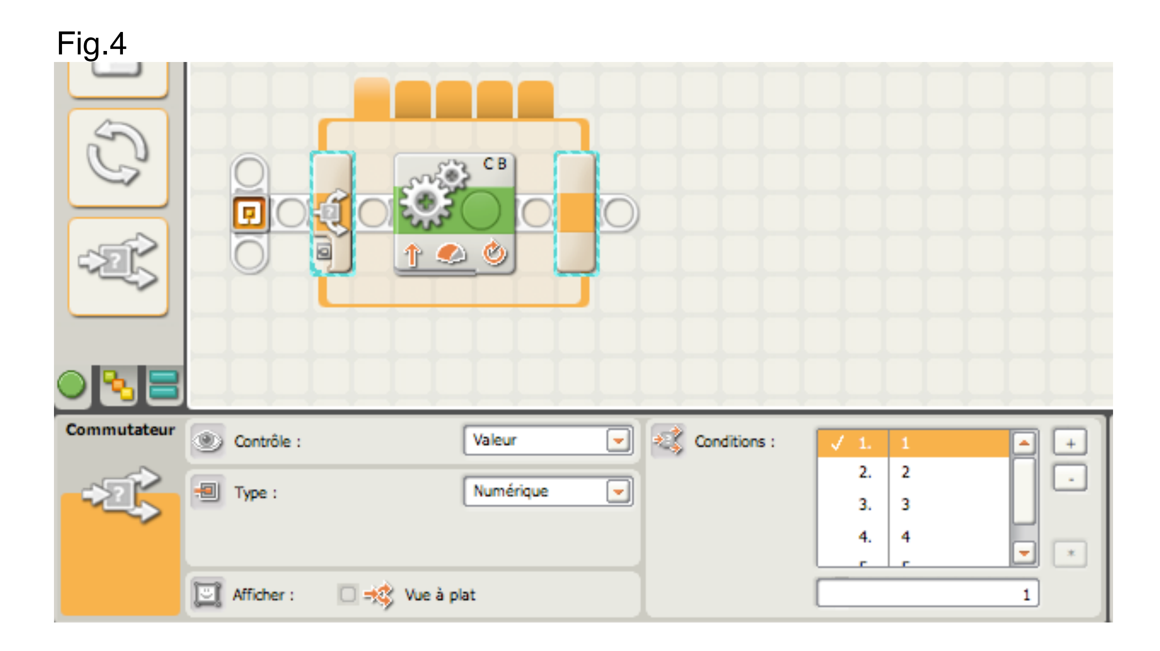
* Dans le premier cas, les 2 chemins sont visibles, le bloc étant fractionné en 2 rayons de séquence, séparés par un trait horizontal. A chacune des parties est associé un symbole: V (pour VRAI) et X (pour FAUX).
On placera les blocs de programmation sur chacun de ces 2 rayons selon les actions envisagées.
La *Vue à plat* (case cochée) est **limitée à 2 chemins uniquement**: L'un est associé au symbole V, l'autre au symbole X. Le symbole V est aussi appelé *chemin de défaut* (mais il peut être modifié).

* Dans le 2ème cas, le fractionnement a disparu et les symbole V ou X sont visibles grâce à des onglets. Chacun rayon de séquence recevra ses blocs de programmation tout en dissimulant l'autre. Il suffira de cliquer sur l'onglet pour faire apparaître le chemin sélectionné. Cet inconvénient est compensé par un grand avantage: celui de définir plus de 2 chemins.

En choisissant le bloc COMMUTATEUR, et selon le nombre de chemins envisagés, vous devrez décocher ou non la *vue à plat.*

Rappelez-vous du pseudo-code précédent où 5 choix étaient envisagés. Voici à quoi ressemblerait le bloc COMMUTATEUR dans ce cas:

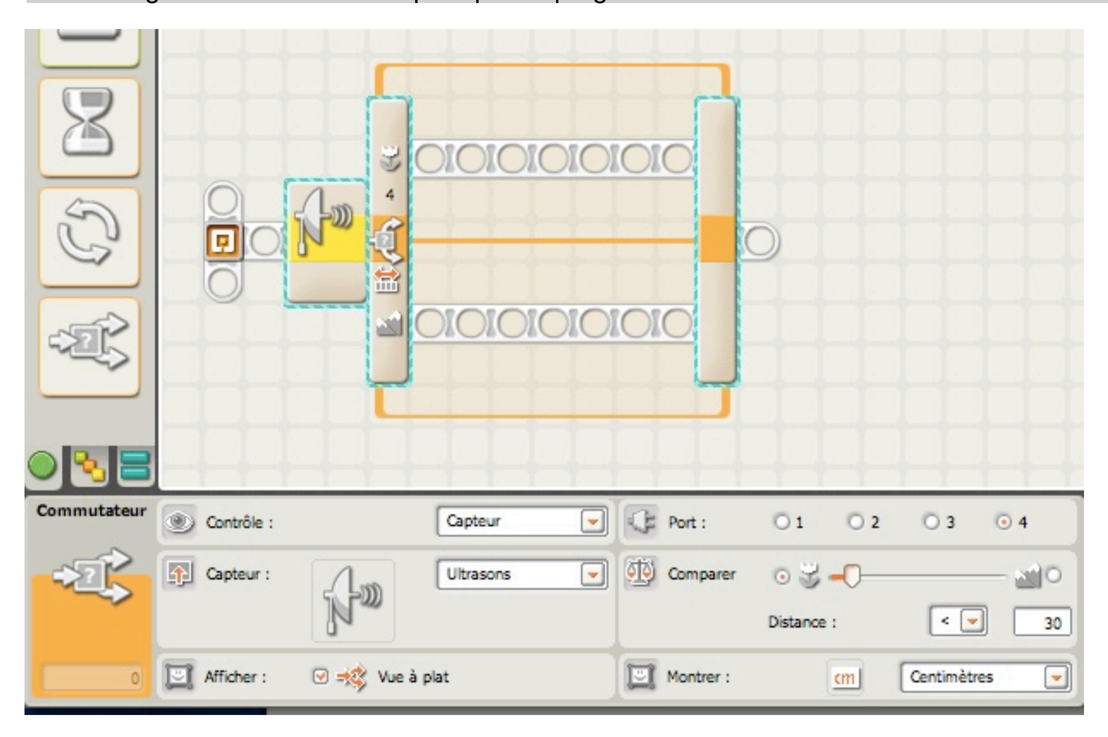
Fig.4



Comme nous sommes en présence de 5 choix, on ne peut plus appliquer le type *Logique* (limité à 2 choix), mais *Numérique*.
Nous verrons plus loin comment utiliser ce bloc pour des chemins plus nombreux que 2.

Je vais à présent vous montrer comment configurer le bloc COMMUTATEUR:
J'ai choisi un simple bloc en mode *Vue à plat*: cela signifie que j'ai seulement 2 chemins possibles pour mon robot. Le premier chemin (symbole V) est sur la partie haute du bloc, le 2ème chemin (symbole X) est sur la partie basse.

Fig.5

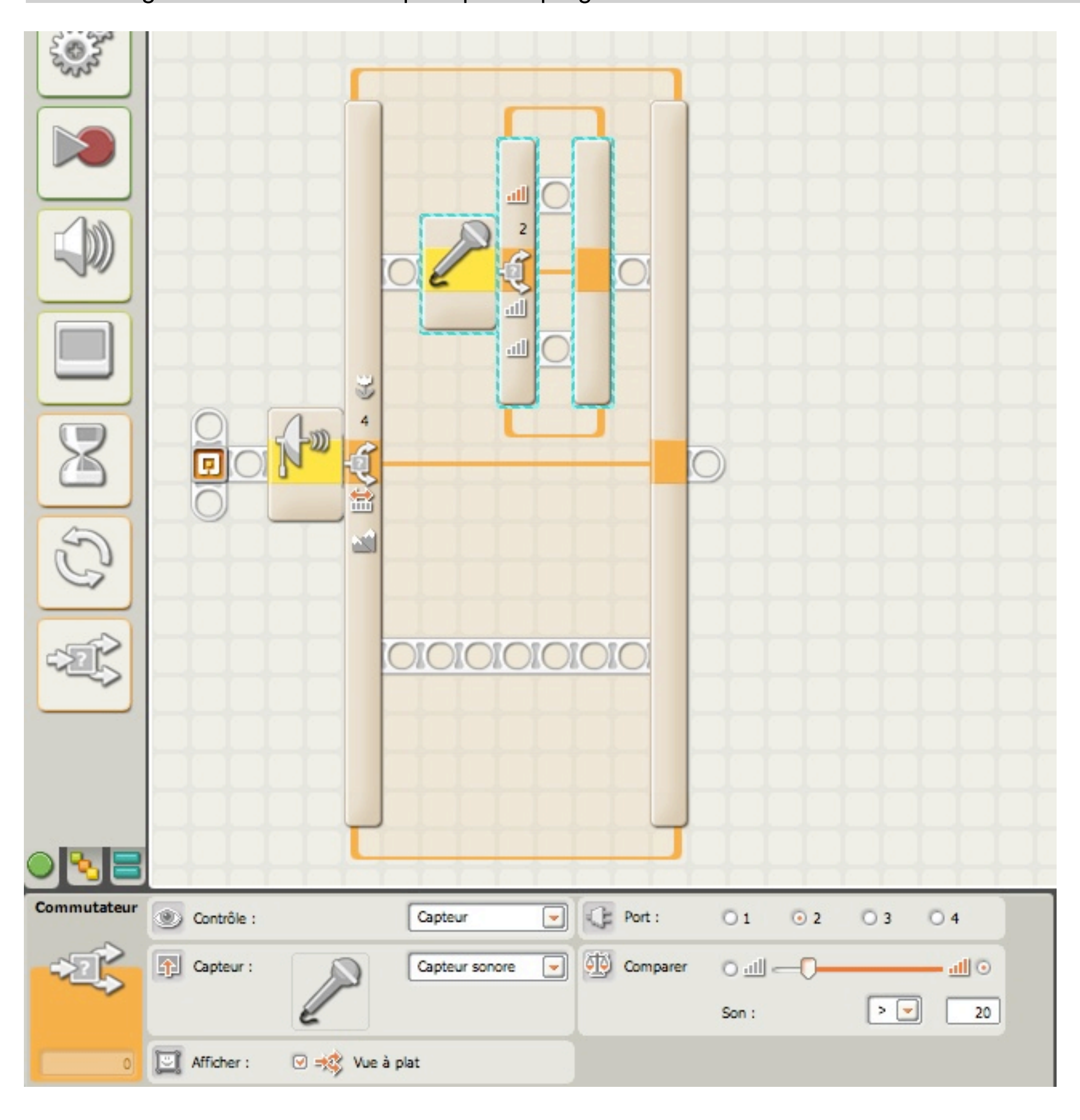


Dans la zone *Contrôle* le champ de saisie offre 2 options: *Capteur* et *Valeur*. J'ai choisi *capteur* puis *Ultrasons*. Les symboles V et X ont disparu, cela signifie que l'option capteur doit préciser lequel des 2 chemins doit correspondre au symbole V (valeur VRAI) quand le point de déclenchement est atteint.
Dans cet exemple j'ai configuré le capteur à ULTRASONS de telle sorte qu'il détecte un objet à moins de 30 cm. Si cette condition est vérifiée, alors le bloc COMMUTATEUR exécutera tous les blocs (s'il y en a) placés sur le chemin VRAI, c'est à dire le chemin du haut. Si cette condition n'est pas vérifiée (distance supérieure à 30 cm), le bloc COMMUTATEUR exécutera tous les blocs (s'il y en a) placés sur le chemin FAUX, c'est à dire chemin du bas.

Je suppose maintenant que BONG dispose d'un capteur à ULTRASONS et un capteur SONORE fixés sur sa structure.
Je lui adresse ce pseudo-code suivant:
LEO > BONG, quand le capteur à ULTRASONS détecte un objet à moins de 30 cm, tourner à gauche *si le capteur SONORE relève un niveau plus grand que 20.*

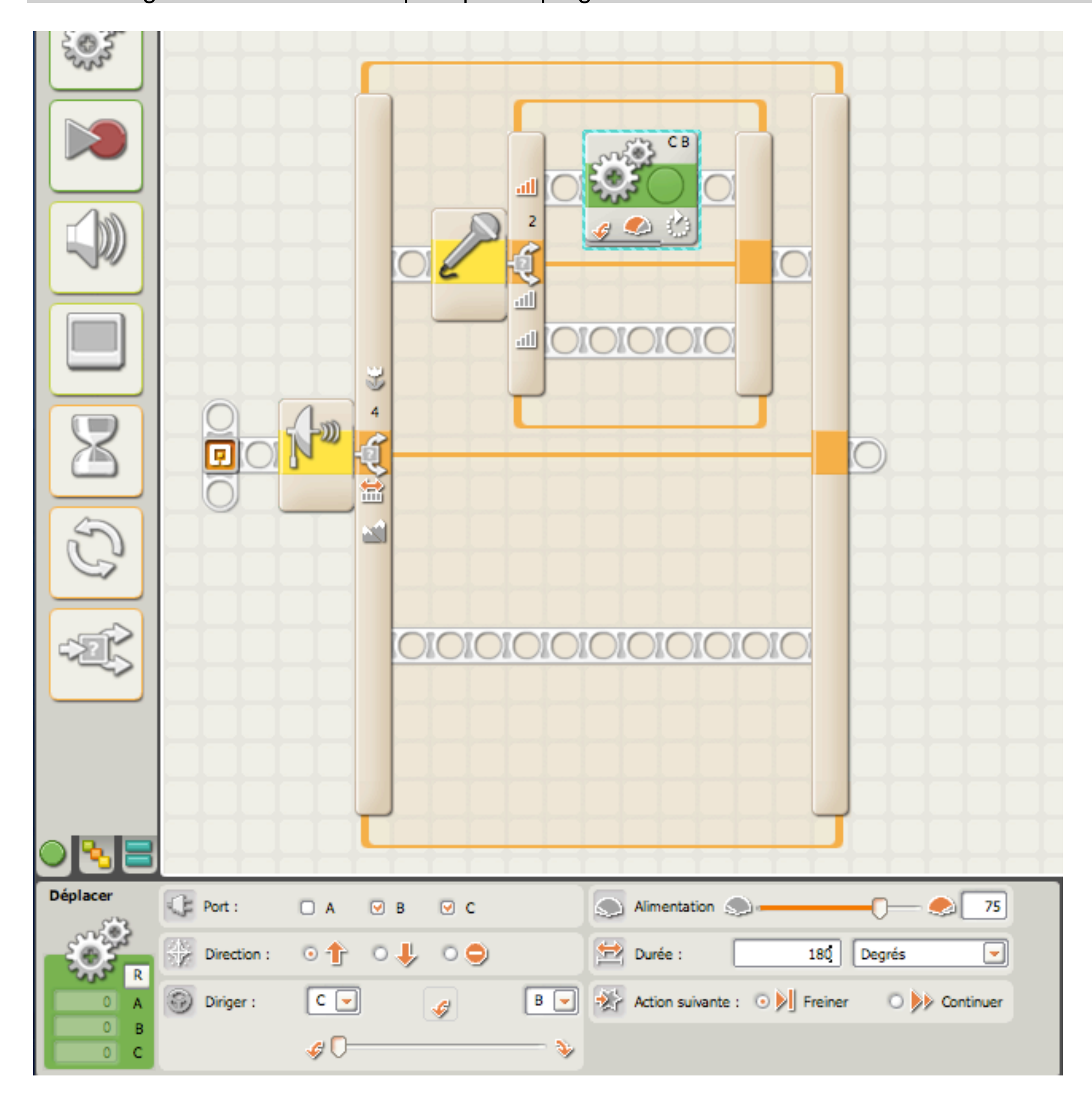
J'ai déjà configuré le premier bloc COMMUTATEUR pour détecter une distance inférieure à 30cm: cela signifie que la suite de mon programme se placera sur le chemin du haut (VRAI), puisque cette première condition est vérifiée. Ensuite je veux que BONG tourne à gauche *si seulement* le capteur SONORE relève un son plus grand que 20. Comment traduire cela? Simplement en utilisant un 2ème bloc COMMUTATEUR.
D'abord je glisse un autre bloc COMMUTATEUR sur le rayon de séquence supérieur ainsi: en prenant le soin de sélectionner par la suite, le bon capteur.

Fig.6



Ensuite, je configure le 2 ème bloc COMMUTATEUR pour un point de déclenchement à 20. Si cette valeur est dépassée, alors la suite du programme (c'est à dire tourner à gauche), trouvera sa place dans le rayon de séquence supérieur de ce 2 ème bloc.

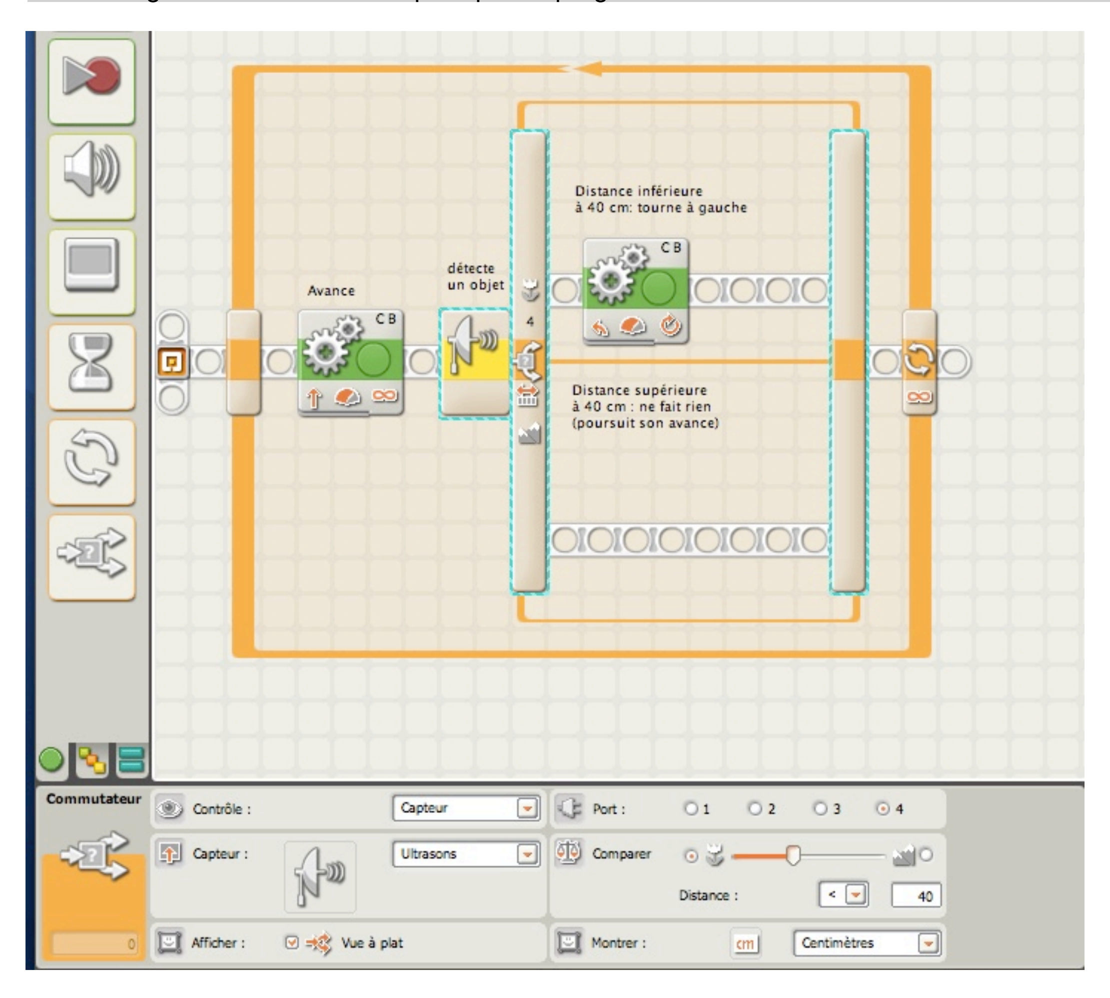
Fig.7



Cet exemple est à l'image de ce que nous avons déjà vu pour les BOUCLES: nous sommes dans un dispositif de blocs COMMUTATEURS IMBRIQUES. En combinant ces blocs, on devine comment donner à BONG la manière de contrôler de complexes décisions.

Un autre exemple combinant le bloc BOUCLE avec celui de COMMUTATEUR:

Fig.8



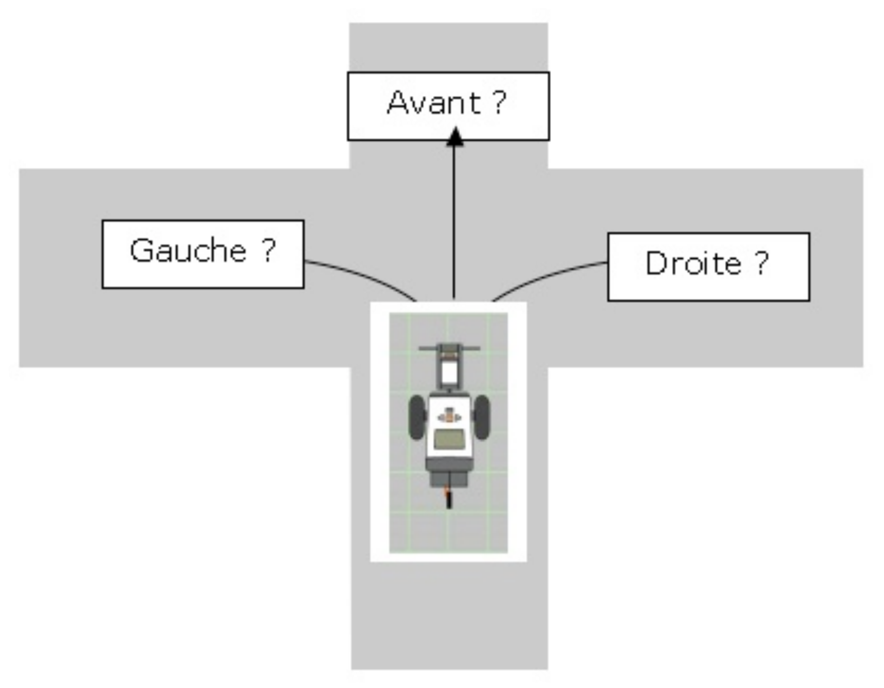
Le premier bloc AVANCER dans la BOUCLE, réglée sur *pour toujours,* provoque le déplacement du robot en droite ligne. Le capteur à ULTRASONS associé au bloc COMMUTATEUR, mesure en permanence la distance entre le robot et un obstacle. Si la distance est inférieure à 40 cm (condition VRAI) il prend le chemin du haut et tourne à gauche. Si la distance est supérieure à 40 cm (chemin du bas), il se contente de ne rien faire et passe la main.
Le bloc COMMUTATEUR ayant accompli sa tâche, on passe au bloc suivant qui est en fait la BOUCLE: on recommence donc le processus.
Le robot va continuer à avancer jusqu'à trouver un nouvel obstacle, et ainsi de suite…
Ce programme très simple permet à un robot d'éviter en permanence un obstacle placé devant lui. Dans cet exemple le robot tourne toujours à gauche; nous verrons plus tard comment lui laisser décider de tourner à gauche ou à droite (2 choix possibles).
Remarque: Le chemin du bas étant de fait vide, il est plus commode pour la lecture du programme de décocher la case *Vue à plat*. Le graphique y gagne en clarté.

Les moteurs relatifs à cet exemple ont été montés à l'envers, d'où le sens inversé de marche. Tenez en compte en cas d'emploi pour vos robots.

Multiples décisions

Voyons à présent comment votre robot peut décider pour un plus grand nombre de choix. Il existe des tas de situations ayant plus de 2 options, non? Aussi, je prendrai un autre exemple pour BONG.

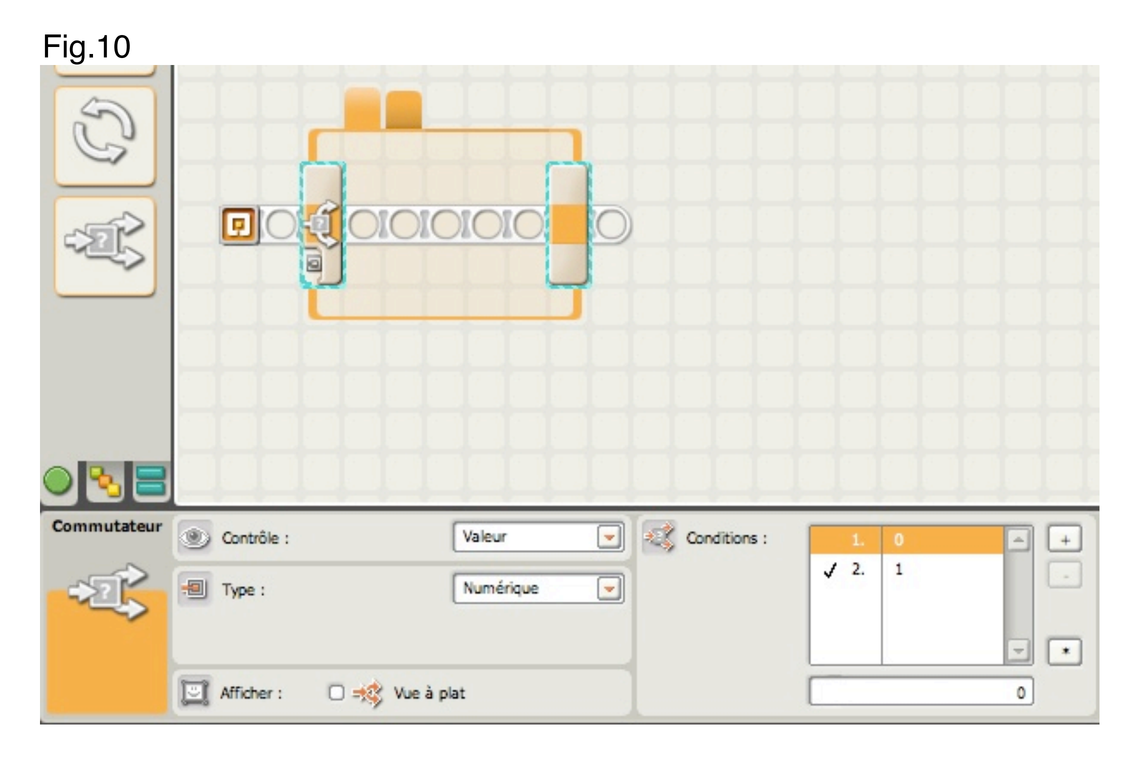
Fig.9



Le *pseudo-code*:
LEO > BONG, en arrivant au croisement, choisir un chiffre entre 1, 2 et 3.
Si le chiffre est 1, tourner à gauche. Si le chiffre est 2, tourner à droite, et si le chiffre est 3 aller tout droit.

En préalable, 2 observations:
Nous avons 3 choix possibles, donc il faut ***décocher*** la case *Vue à plat*.
Ensuite, pour configurer le bloc COMMUTATEUR à 3 chemins, il faut choisir dans le panneau de configuration *Valeur* et *Numérique*.

Fig.10

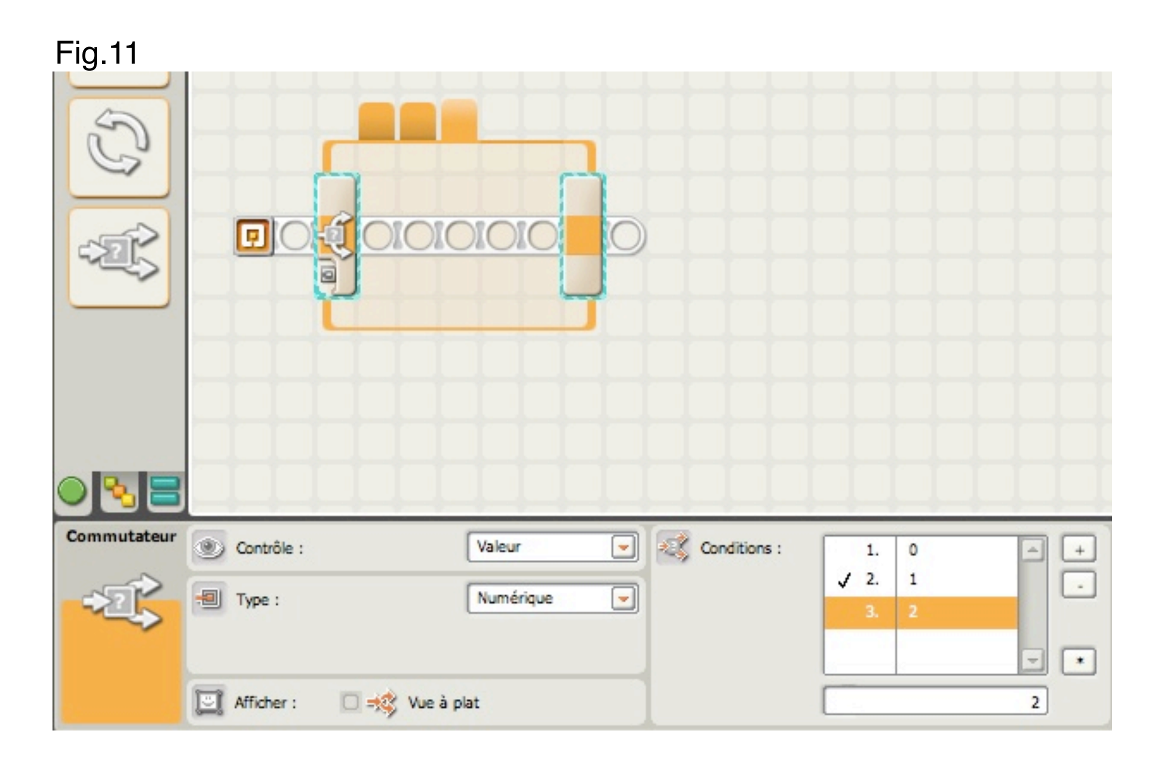


Vous remarquerez que le bloc COMMUTATEUR dispose de 2 onglets et d'un plot de données en entrée que nous allons utiliser. Ce plot accepte les 3 types de données déjà

évoqués au cours des dernières leçons. Pour la cause, j'ai choisi *Numérique*, parce que des nombres caractériseront les différents chemins. Evident, non?
Observez maintenant dans la zone Conditions ce petit tableau comprenant 2 lignes et 2 colonnes. La première colonne contient le numéro du chemin. La 2 ème colonne contient une valeur de défaut. Ici, pour le chemin 1, c'est 0, et pour le chemin 2, c'est 1.
Donc, si par le plot de donnée j'entre une valeur 0, le chemin n° 1 sera sélectionné. Si j'entre la valeur 1, c'est le chemin n° 2 qui sera sélectionné.

Vous suivez? …Bien.
Or, mes besoins m'imposent 3 chemins. Il faudra donc que j'ajoute un chemin supplémentaire. Pour cela, appuyez sur la case + (*Ajouter une nouvelle condition*), en haut et à droite du tableau.

Fig.11



J'ai à présent un 3 ème chemin avec une valeur de défaut de 2.

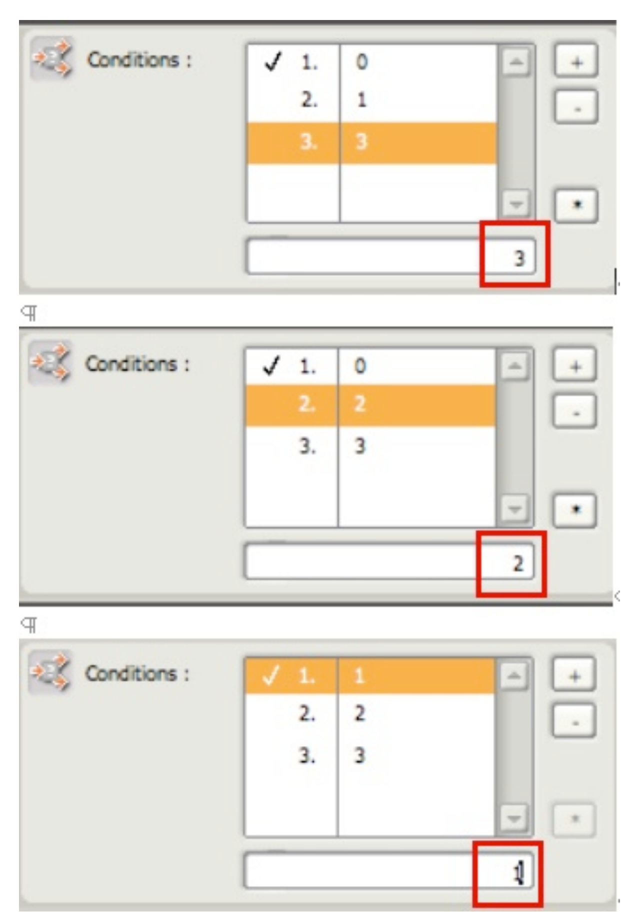
Les valeurs numériques que j'ai décidé d'attribuer à chaque chemin le seront d'une manière aléatoire, grâce à un bloc particulier que nous verrons tout à l'heure.
Pour le moment, il s'agit pour chacun des chemins de leur attribuer une valeur de défaut correspondant à leur numéro: 1 pour le chemin n°1, 2 pour le chemin n°2 et 3 pour le chemin n°3.

Le tableau se présente actuellement ainsi:

1.	0
2.	1
3.	2

Nous allons le modifier de la manière suivante:

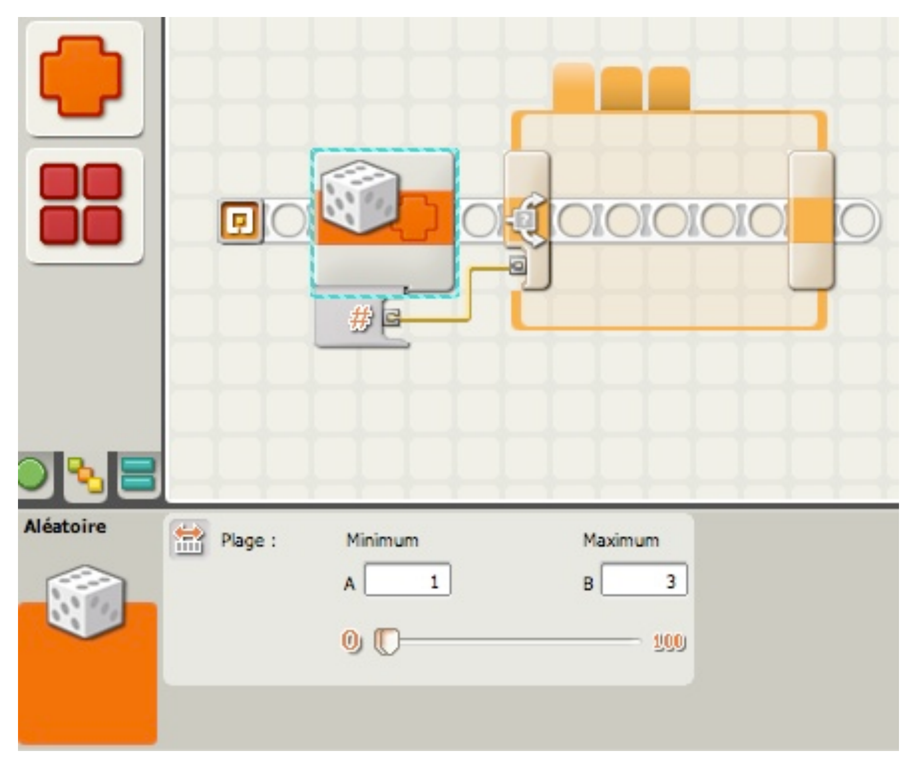
Fig.12



Chaque ligne étant sélectionnée (en orange), on portera successivement es valeurs dans le champ de saisie au dessous.
Cette modification a fait disparaître la valeur zéro, nous verrons pourquoi.
On remarquera que la valeur de défaut est cochée en ligne n°1; cela signifie que le 1er onglet correspond au chemin n°1.

Je vais à présent introduire un nouveau bloc ALEATOIRE devant le bloc COMMUTATEUR. Ce bloc est un bloc mathématique qui ne sert qu'à générer une valeur quelconque, au hasard, comprise entre 2 bornes (maximum et minimum) qui en font aussi partie. Par défaut, ces bornes sont 0 et 100. Nous allons les remplacer par 1 et 3. Ensuite, nous tirerons un fil de données, pour obtenir le résultat suivant:

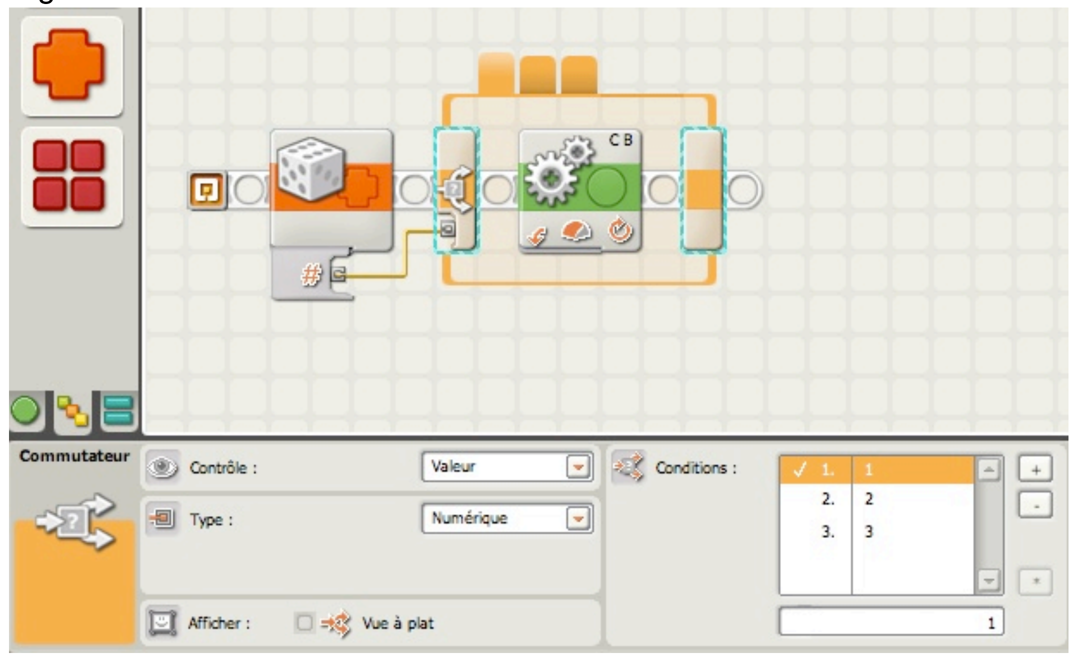
Fig.13



Ce bloc a la particularité de produire une valeur 1, 2 ou 3, chaque fois qu'il sera sollicité. Le zéro ayant été éliminé, c'est la raison pour laquelle il ne doit pas se retrouver dans la valeur du chemin n°1.
On a rendu cohérent les n° de chemin avec leurs valeurs associées

Et maintenant, on peut insérer les blocs DEPLACER dans chaque chemin en fonction de l'ordre des onglets:
Onglet 1: tourner à gauche,
Onglet 2: tourner à droite
Onglet 3: tout droit

Fig.14

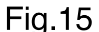


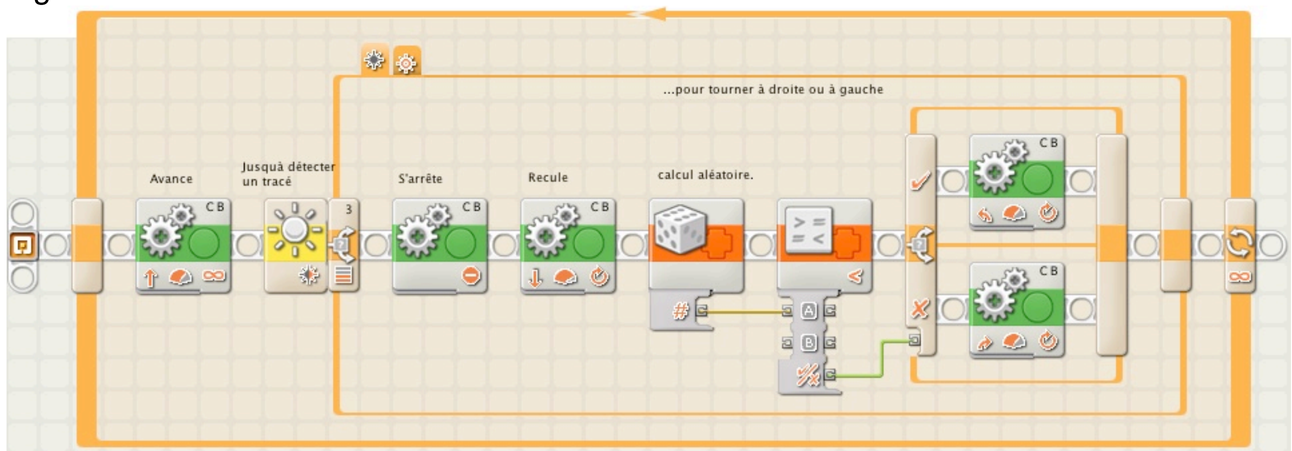
Et si on souhaite que l'opération se répète, il suffira de placer l'ensemble dans une BOUCLE *pour toujours*.

A chaque passage de la boucle, le bloc ALEATOIRE produira au hasard un nombre 1, 2 ou 3 qui sera comparé aux conditions du bloc COMMUTATEUR, lequel ouvrira la chemin du même n°.

Simple, élégant et sans fin!
Et que se passerait-il si le bloc ALEATOIRE devenait fou et produisait une valeur 4?
Cette valeur n'existant pas dans les conditions, c'est la valeur de défaut qui serait alors sollicitée; dans ce cas le chemin n°1 (valeur cochée). De plus quelle que soit la valeur sauf 2 ou 3, c'est toujours ce chemin n°1 qui sera choisi.

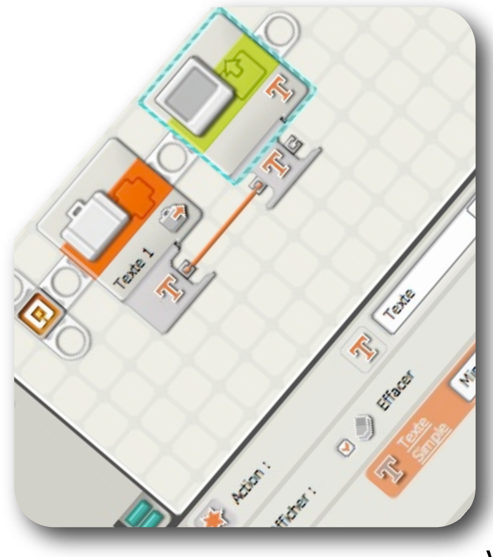
Enfin et pour terminer:
Encore un autre exemple pour se faire plaisir:
Voulez-vous faire évoluer un robot à l'intérieur d'un tracé fermé? Essayez alors ce programme: Un bloc BOUCLE avec 2 blocs COMMUTATEUR IMBRIQUEES.


Fig.15



Le bloc ALEATOIRE est utilisé ici pour choisir au hasard une valeur comprise entre 0 et 100. Cette valeur est ensuite comparée à un valeur constante B = 25. Selon le résultat qui est une valeur logique (VRAI ou FAUX), le robot empruntera le chemin du haut (tourne à gauche) ou celui du bas (tourne à droite).

Retenez que le bloc COMMUTATEUR est un bloc puissant qui donne énormément de possibilités de décisions selon le type de données en entrées, et les capteurs associés.



Leçon n°9: VARIABLE....la VALISE FOURRE-TOUT....

Drôle te titre pour une leçon! Cela signifie-t-il que l'on peut mettre n'importe quoi dans une valise, mais pour faire quoi?
En vérité cette leçon traite un bloc spécial appelé bloc VARIABLE. Vous trouverez ce bloc très utile lorsque vous aurez besoin de ranger une information pour un usage ultérieur.

Le bloc VARIABLE

Imaginons un moment que vous voulez donner à BONG quelques informations à retenir. Ces informations peuvent consister en quelques mots, quelques nombres, et quelques couples de valeurs logiques OUI/NON.
BONG a la faculté de ranger chaque morceau d'information dans un dossier virtuel existant dans sa mémoire.

Imaginons ce *pseudo-code*:
LEO > BONG Voulez-vous ranger les mots "roue" et "engrenage" dans votre mémoire?
LEO> BONG, j'ai aussi besoin que vous rangiez les nombres "120" et "70" dans votre mémoire"
LEO> BONG voulez-vous ranger un "VRAI" logique et un "FAUX" logique dans votre mémoire?

Maintenant, avant de traduire ce *pseudo-code* en programme NXT-G, il me faut vous éclairer un peu plus sur la manière dont un programme NXT-G range une information, la modifie, et la retrouve. Tout cela est réalisé par le bloc VARIABLE.

Un bloc VARIABLE peut accomplir qu'un des 2 choses suivantes:

Une information peut-être *écrite* dans un dossier virtuel qui est rangé en mémoire.
Une information peut-être *lue* à partir d'un dossier virtuel rangé en mémoire.

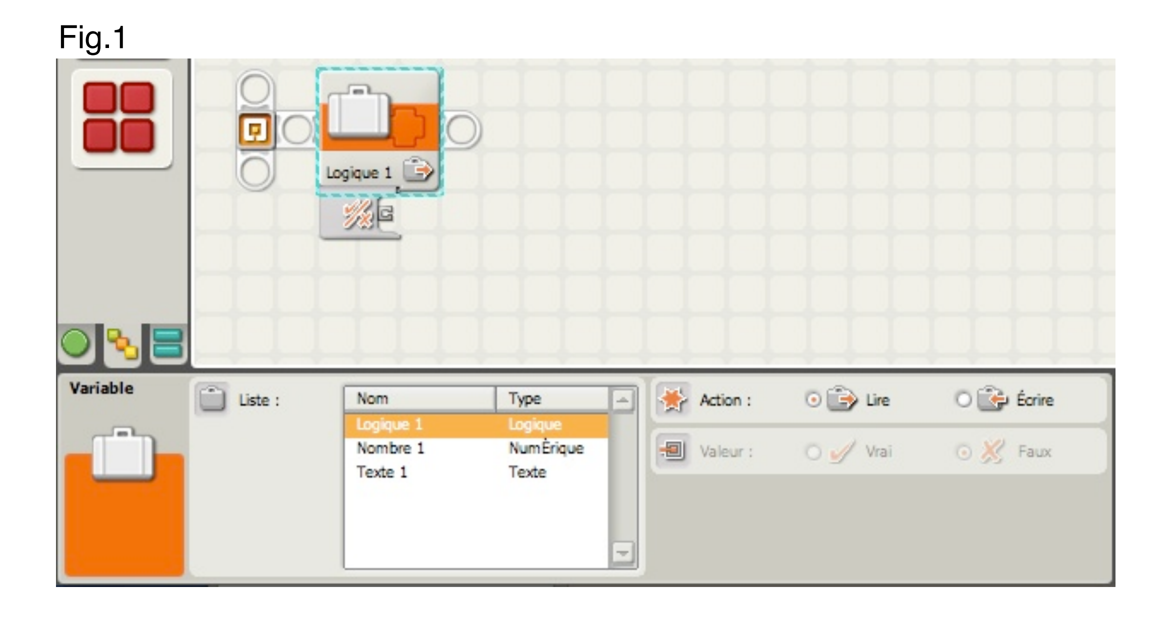
Ces dossiers virtuels sont généralement connus sous le nom de *variables*.
Une variable NXT-G ne peut recevoir que 3 sortes de données:
Texte, Nombre, et valeur Logique (OUI/NON)
Vous avez déjà vu cela, n'est-pas? Et bien c'est la même chose pour les variables.

Un Texte est facile à comprendre; LEO a demandé à BONG de ranger "roue" et "engrenage"; il aurait été aussi simple de lui dire de ranger la lettre "A" ou la phase "Mon nom est BONG" .

Pour les Nombres, c'est un peu différent: quand un NXT-G bloc VARIABLE est paramétré pour recevoir un nombre, *ce ne peut-être qu'un <u>nombre entier</u>*, positif ou négatif. Les nombres comme 8,5 ou -16,7 seront arrondis à l'entier le plus proche (8 et -17 ici).
Les valeurs Logiques n'ont que 2 choix: VRAI ou FAUX. Un bloc VARIABLE NXT-G, paramétré pour recevoir une valeur Logique, *ne peut contenir seulement* que VRAI ou FAUX, et rien d'autre.

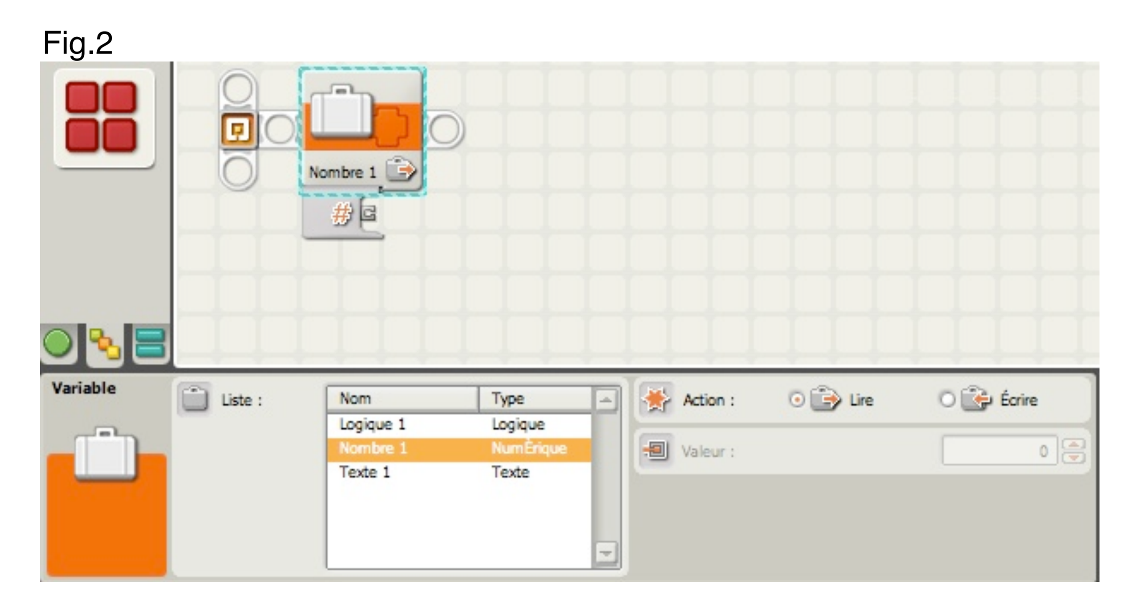
Bien, voyons à présent à quoi ressemble le bloc VARIABLE.
Il est situé dans la palette *Complète* au développé de l'icône "Données".

Fig.1



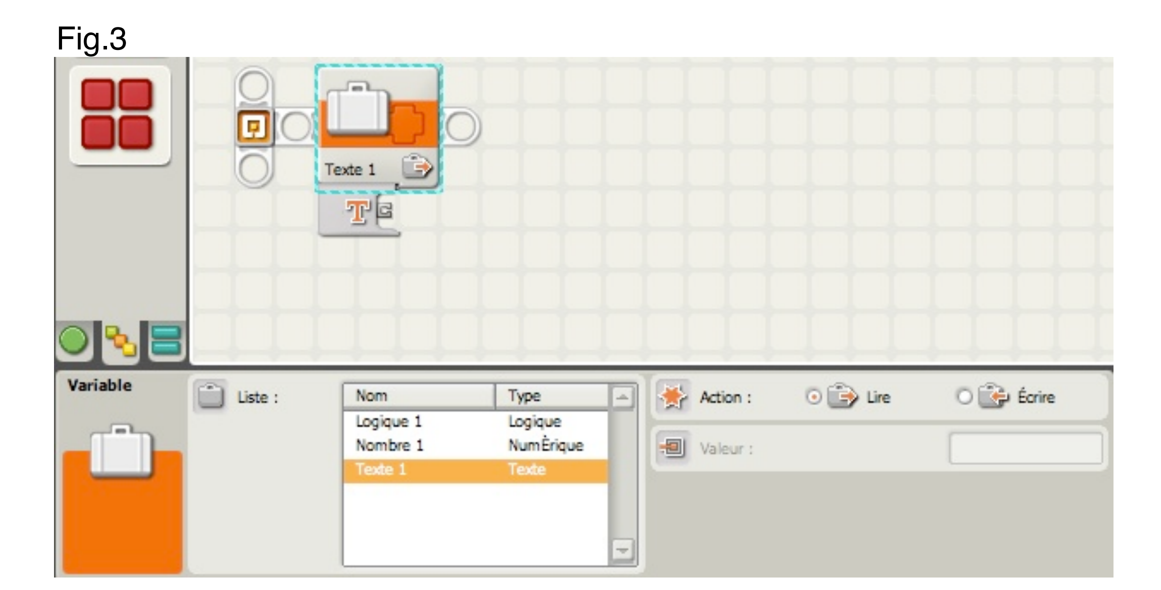
J'ai signalé plus haut qu'une valeur pouvait être *écrite* dans ou *lue* dans un bloc VARIABLE.
Ici, par défaut le bloc est configuré pour lire (dans la zone Action, lire est sélectionné) une valeur Logique. Cette variable a aussi un nom: Logique 1.
Cela signifie que si VRAI ou FAUX est rangé dans la variable, cette valeur ne peut-être que ***lue***. Vous remarquerez aussi que la zone Valeur est estompée donc inaccessible.
Enfin, toujours dans cette zone estompée la valeur de défaut est FAUX.
Avant de vous montrer comment modifier ces paramètres, sélectionnez la variable Nombre 1.

Fig.2



Vous remarquerez que la valeur de défaut (dans la zone *Valeur* estompée) est zéro (0).
Ce nombre est la valeur qui sera lue dans la Variable "Nombre 1".
Enfin, choisissez la variable Texte 1.

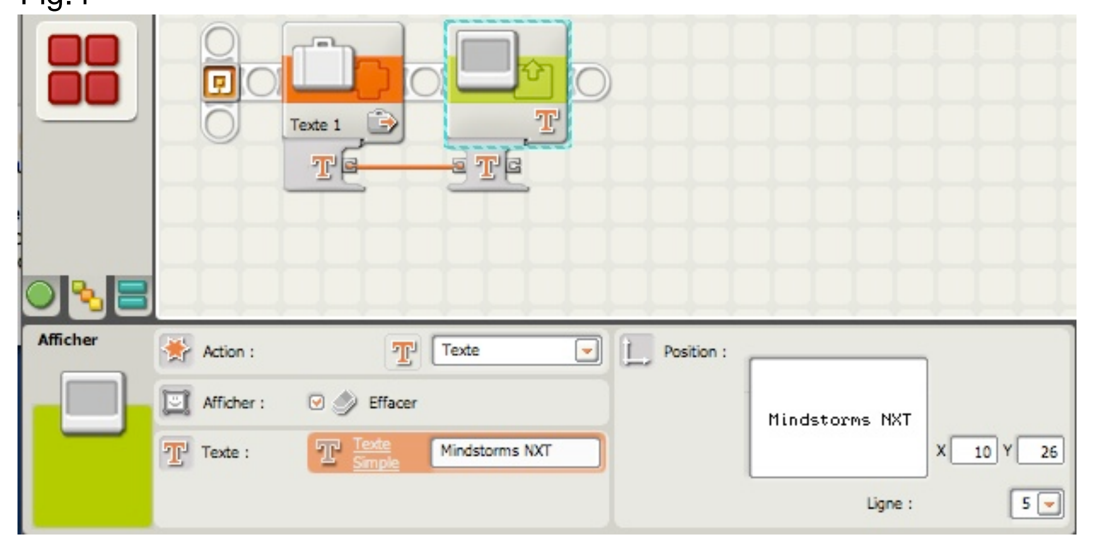
Fig.3



La valeur de défaut (dans la zone *Valeur* estompée) dans le champ de saisie est vide; il n'y a aucun texte rangé dans cette variable.
Dans les 3 figures ci-dessus, on notera que les blocs VARIABLE ne disposent que d'un seul plot de données, le plot de Sortie; évident, non?
Dans ces 3 cas, les données ne peuvent-être que lues. Et pour les lire j'ai besoin d'un autre bloc disposant lui aussi d'un plot de données, par exemple le bloc AFFICHER. Nous savons que ce bloc dispose d'un plot d'ENTREE "Texte" (T); il suffira donc de tirer un fil de données entre ces 2 blocs.

Fig.4

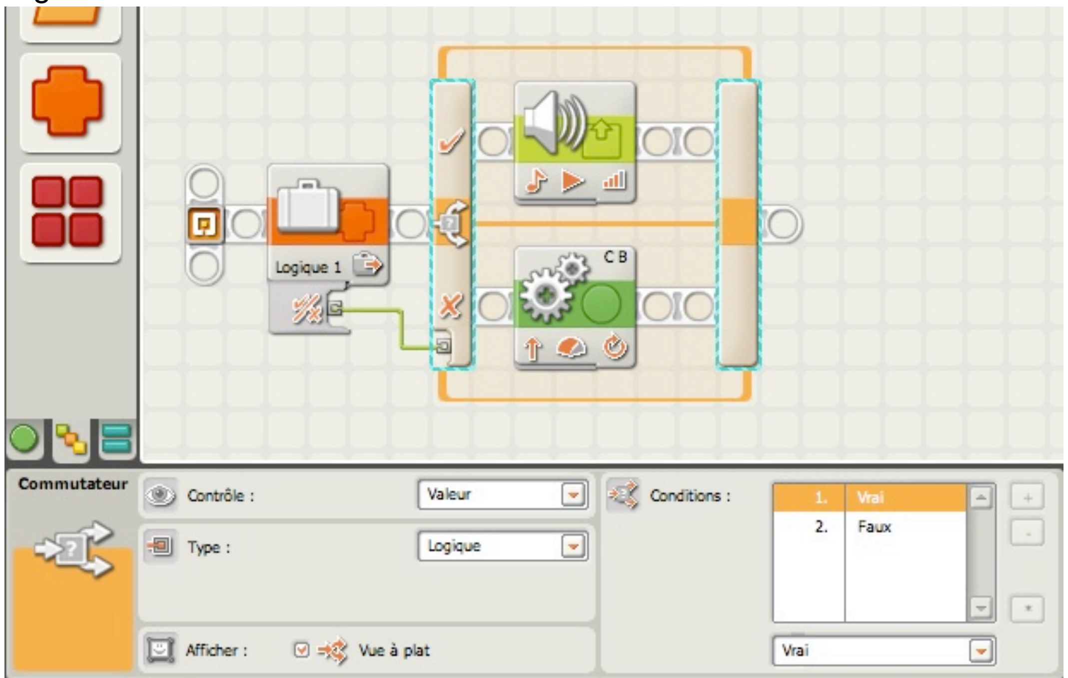




Le texte figurant dans la zone Texte du bloc AFFICHER (Mindstorms NXT) n'apparaîtra pas sur l'écran, mais sera remplacé par la valeur de la variable (ici néant). Par contre si un texte figurait dans la zone valeur de la variable, c'est alors ce texte qui aurait pris la place, sur la ligne 5, bien évidemment.

J'espère que vous avez compris ce mécanisme, et je vais maintenant l'illustrer d'une autre manière.

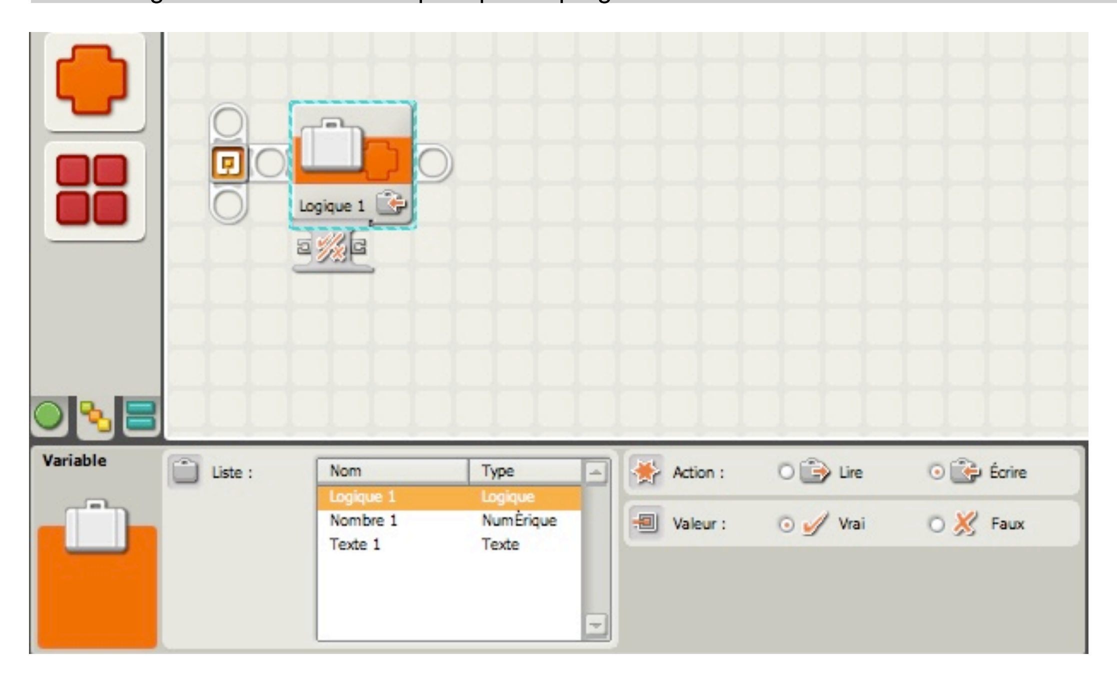
Fig.4



Dans cet exemple, j'ai connecté un bloc VARIABLE nommé Logique 1 à un bloc COMMUTATEUR dont le panneau de configuration est visible sur la figure. Ces 2 blocs sont compatibles puisqu'ils n'acceptent que des valeurs logiques.
J'ai relié les plots par un fil de données. La valeur de défaut du bloc VARIABLE est FAUX. Quand j'exécute le programme, le bloc COMMUTATEUR lira le contenu du bloc VARIABLE. Si la valeur est VRAI, le bloc COMMUTATEUR activera le bloc SON qui se trouve dans la partie haute (VRAI); si la valeur est FAUX, le bloc COMMUTATEUR activera le bloc DEPLACER qui se trouve dans la partie basse (FAUX).

Après avoir traité le bloc VARIABLE en *lecture*, que se passerait-il si je voulais y introduire des données?
Pour cela je dois *écrire* des données dans ce bloc, et voici comment s'y prendre.

Fig.5



Après avoir placé un bloc VARIABLE sur le rayon de séquence, j'ai choisi Logique 1 comme valeur.
Dans la zone *Action*, j'ai sélectionné *Ecrire*. La zone *Valeur* devient maintenant accessible, et là j'ai sélectionné VRAI. J'ai donc inversé les valeurs.
Vous avez également remarqué l'apparition d'un plot d'ENTREE. Cela signifie que peux disposer d'une valeur FAUX ou VRAI par l'intermédiaire d'un fil de données relié à un autre bloc. Dans cet exemple, les données doivent être compatibles, sinon le lien sera rompu et représenté par un fil pointillé.
Dans la figure suivante, vous pouvez voir à quoi ressemblent les panneaux de configuration pour le Nombre 1 et le texte 1.

Fig.6



Pour la variable Nombre 1, les valeurs acceptées dans le champ *valeur* sont des **nombres entiers**, positifs ou négatifs.
Pour la variable Texte 1, vous pouvez taper des mots, des phrases ou des paragraphes entiers dans le champ de saisie *valeur*.
Après avoir modifié les données dans un bloc VARIABLE, vous pouvez, soit conserver la variable en sélectionnant lire dans la zone *Action* (cela protège les données de toute fausse manoeuvre), soit quitter le bloc en l'état. Si vous choisissez Ecrire dans la zone Action, vous pouvez alors tirer un fil de données entre un autre bloc et le plot d'ENTREE du bloc VARIABLE (veillez à choisir le type de données compatible avec le bloc pourvoyeur de données - logique, nombre ou texte -).

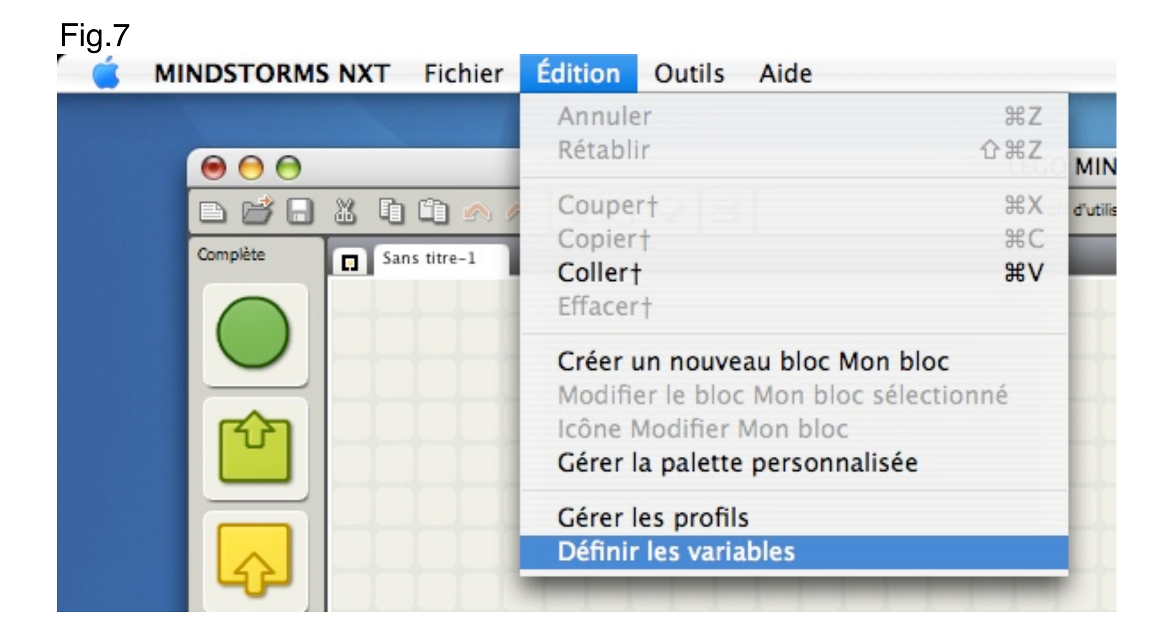
Revenons maintenant a notre *pseudo-code* du début:
LEO > BONG Voulez-vous ranger les mots "roue" et "engrenage" dans votre mémoire?
LEO> BONG, j'ai aussi besoin que vous rangiez les nombres "120" et "70" dans votre mémoire"
LEO> BONG voulez-vous ranger un "VRAI" logique et un "FAUX" logique dans votre mémoire?

Voyez-vous le problème?
J'ai 2 zones de texte et 2 valeurs à stocker. Je sais déjà qu'un bloc VARIABLE ne peut ranger ***qu'une seule*** donnée. Si je range le nombre 120 par exemple dans la variable Nombre 1, où vais-je ranger la valeur 70? Je peux aussi ranger "roue" dans la variable Texte 1, mais qu'en est-il d' "engrenage"?
La réponse est simple. NXT-G vous permet de créer autant de variables que vous le souhaitez. Vous pouvez même renommer la variable pour la rendre plus explicite; remplacer Texte 1 par "Pièce", ou Nombre 1 par "Quantité".
Voilà comment procéder.

Définir les variables

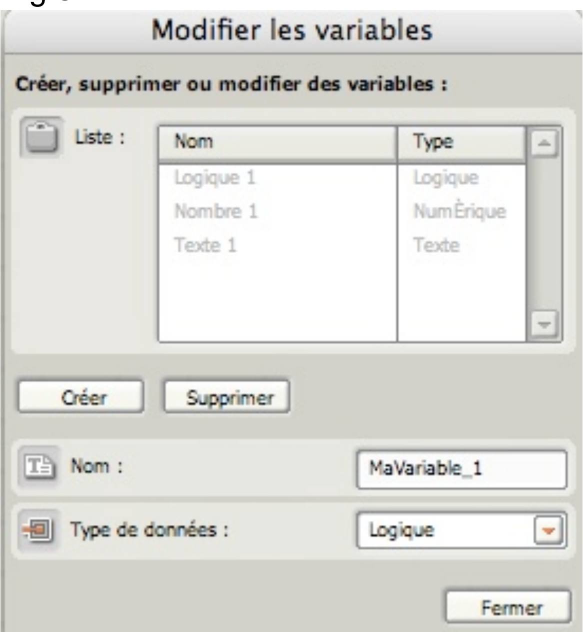
Si vous devez créer une variable supplémentaire, commencez par dérouler dans la barre de menu de la fenêtre du logiciel, le menu EDITION, puis le sous menu DEFINIR les VARIABLES.

Fig.7



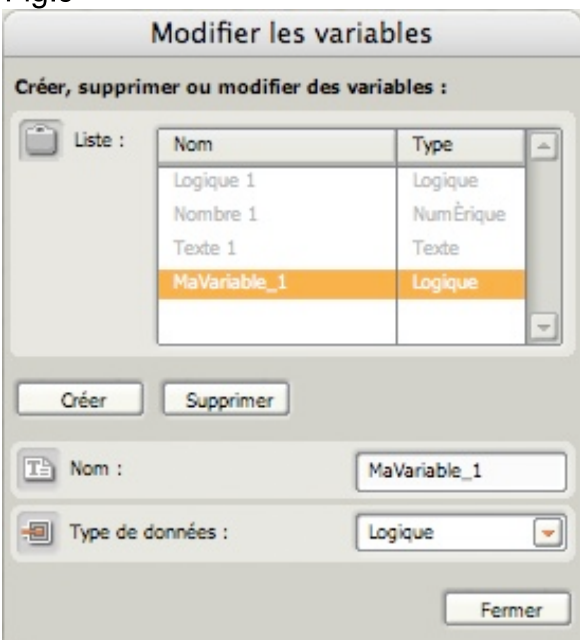
La petite fenêtre suivante apparaît:

Fig.8



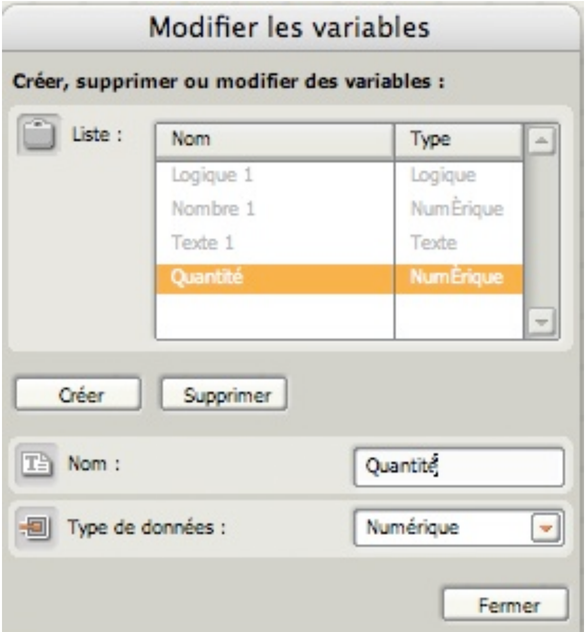
Appuyez sur le bouton ***Créer***.

Fig.9



Une 4 ème variable apparaît sur un bandeau orange; il suffit de modifier le nom de défaut "MaVariable 1" par une autre expression dans le champ *Nom*, et de sélectionner le *type* à utiliser.

Fig.10

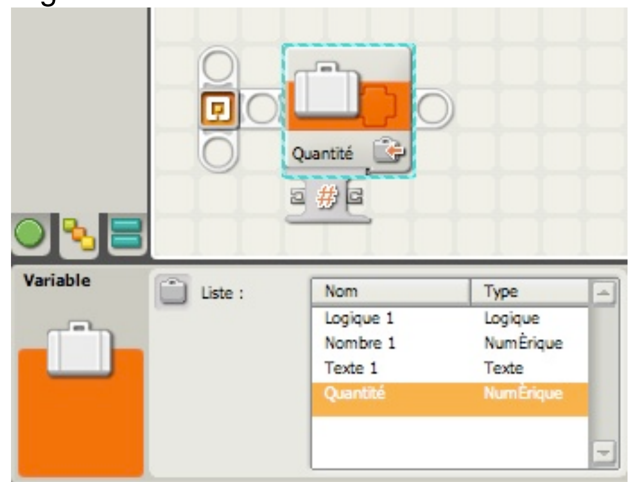


Cliquez sur le bouton ***Fermer.***

Vous venez de créer une nouvelle variable qui vient s'ajouter à celles existantes.

Si vous sélectionnez à présent dans la palette Complète le bloc VARIABLE, voici ce que verrez:

Fig.11



Vous avez créé une variable numérique intitulée *Quantité*, et son nom apparaît dans l'icône du bloc. C'est une manière d'éviter aussi toute confusion dans leur utilisation et leur lecture dans les programmes.
Remarque: cette nouvelle variable n'existe que dans le programme en cours.

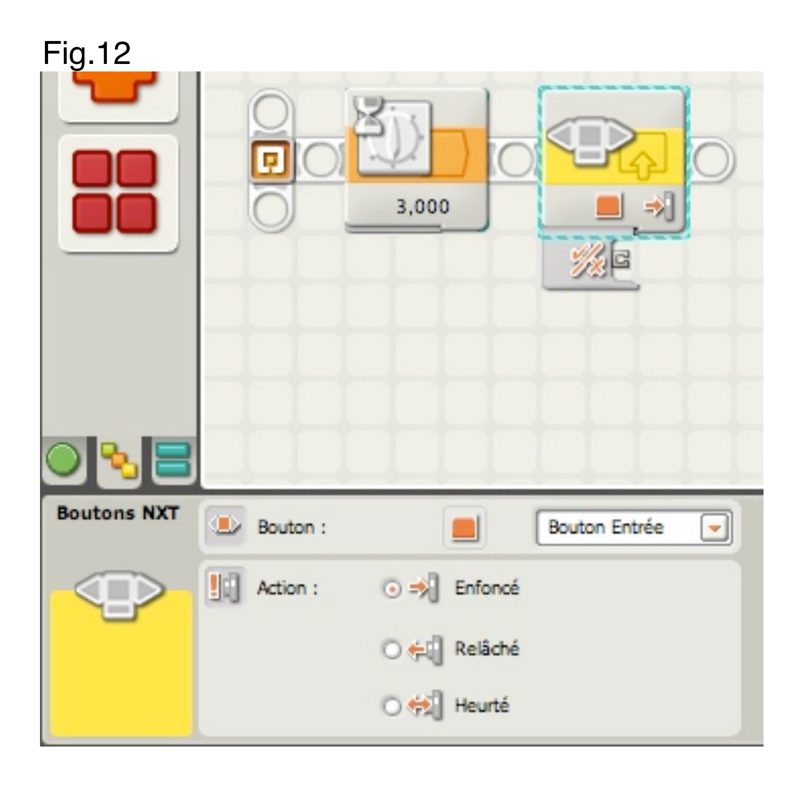
Bien, vous savez maintenant comment écrire et lire des données dans une variable, ainsi qu'en ajouter.

Nous allons aborder la façon de les utiliser dans vos programmes.
Prenons le cas de ce morceau de *pseudo-code* pour BONG:

LEO > BONG, attendre 3 secondes avant que j'appuie sur le le bouton rouge (Entrée).
LEO > BONG, si j'appuie et relâche le bouton, affiche sur ton écran "Enfoncé"
LEO > Bong, si je n'appuie pas sur le bouton, affiche sur ton écran "Lâché".

Je commencerai par installer ces deux blocs:
Un bloc ATTENDRE réglé sur 3 secondes
Un bloc BOUTONS NXT configuré sur Enfoncé.

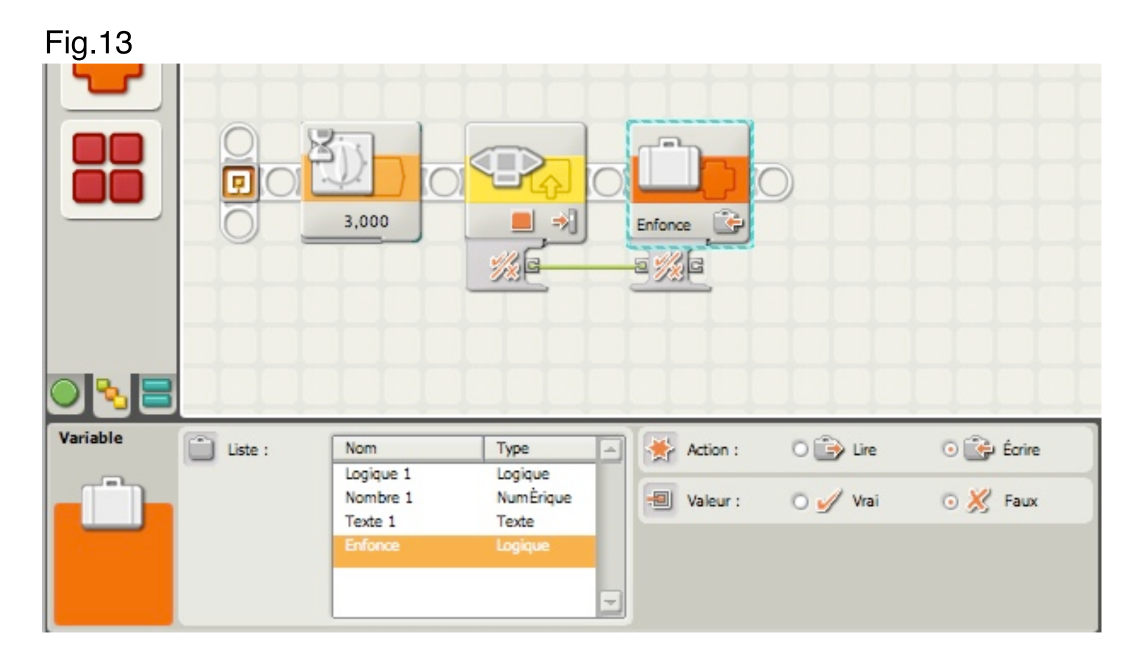
Fig.12



Je vais maintenant créer une variable que j'appellerai *Enfoncé* du type *Logique* (vous savez comment faire…).

J'installe donc à la suite ce bloc VARIABLE, et dans la zone Action, je sélectionne *Ecrire*. On constate l'apparition du plot d'entrée de données sur la gauche de l'icône. Je tire ensuite un fil de données entre le bloc BOUTONS NXT et le bloc VARIABLE.

Fig.13

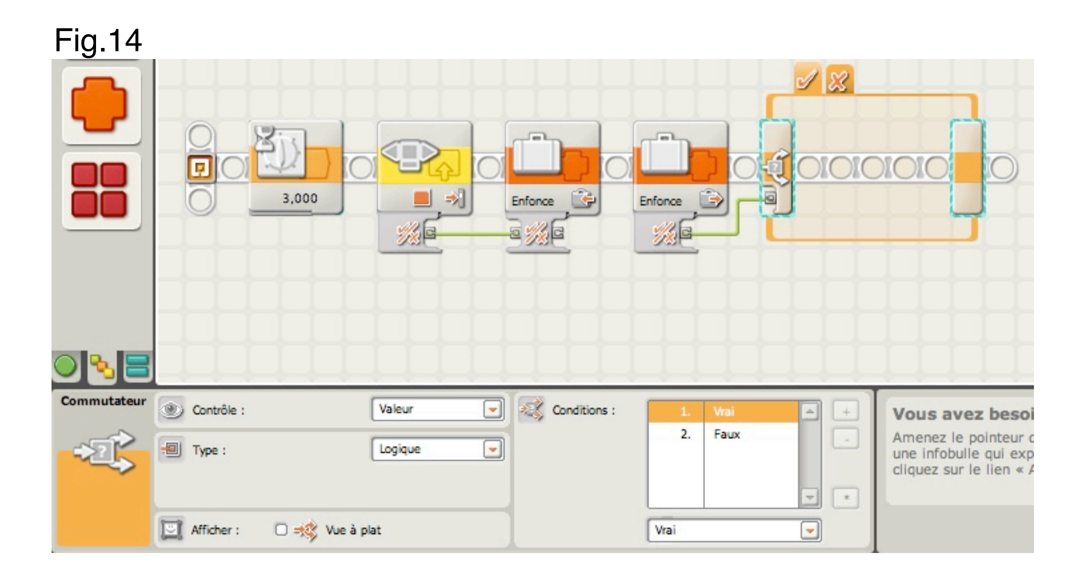


Si le bouton ENTREE du NXT est appuyé, puis relâché **pendant les 3 premières secondes,** le bloc capteur boutons NXT détectera le changement d'état du bouton et

intervertira la valeur logique FAUX par VRAI. Une valeur VRAI ou FAUX sera alors écrite dans la variable ENFONCE jusqu'à ce que les 3 secondes soient écoulées.

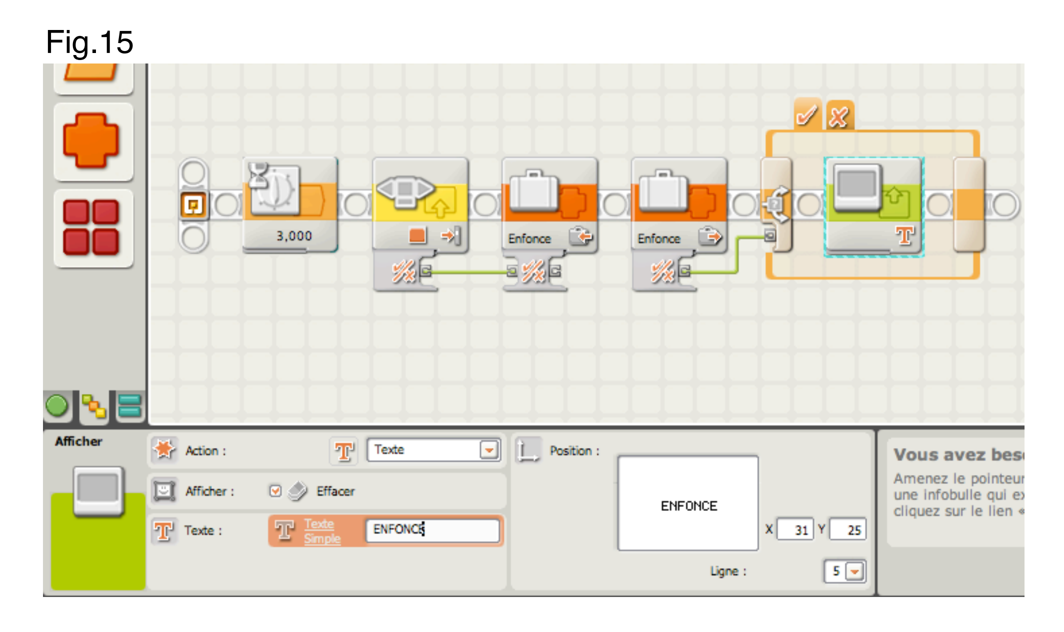
Au démarrage du programme, la valeur initiale est FAUX, mais elle peut être changée en VRAI si le bouton ENTREE est enfoncé.
Lorsque le bloc ATTENDRE se termine (après 3 secondes), il faut afficher sur l'écran du NXT "Enfoncé" ou "Lâché". Pour accomplir cette tâche, je vais ajouter un autre bloc variable ENFONCE en prenant le soin de sélectionner dans la zone *Action*, l'option *Lire*. Ensuite je vais lui associer un bloc COMMUTATEUR paramétré en valeur-logique, avec une représentation en onglets (case *Vue à plat* décochée). Pour compléter je tire un fil de données entre cette nouvelle variable et le bloc COMMUTATEUR.
Ce qui donne la figure suivante:

Fig.14



Le bloc COMMUTATEUR réagira selon la valeur résidant dans la variable ENFONCE. Dans le chemin VRAI du bloc COMMUTATEUR, je place un bloc AFFICHER comportant le mot "ENFONCE", et dans le chemin FAUX, un autre bloc AFFICHER comportant le mot "LACHE" (la case *Effacer* étant cochée dans les 2 cas).

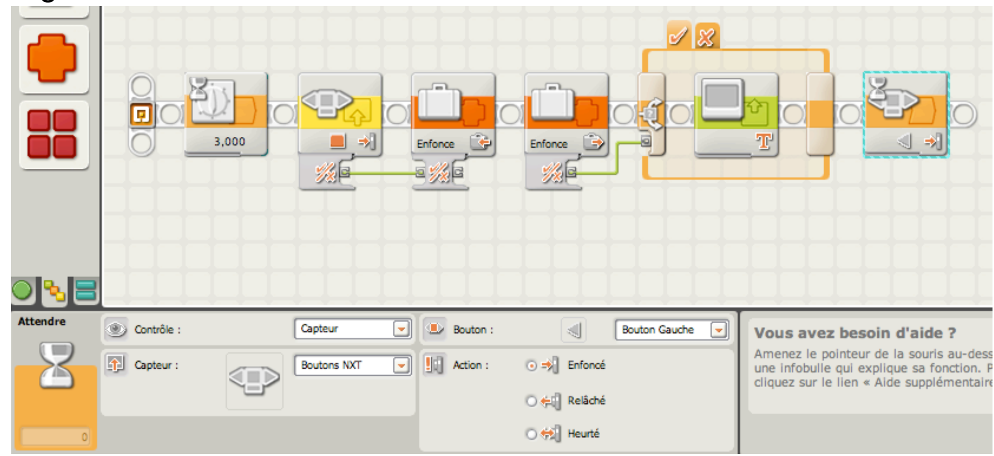
Fig.15



Enfin et pour terminer le programme, j'installe un bloc ATTENDRE jusqu'à ce que le bouton flèche Gauche du NXT soit enfoncé.

Ce qui donne le résultat final:

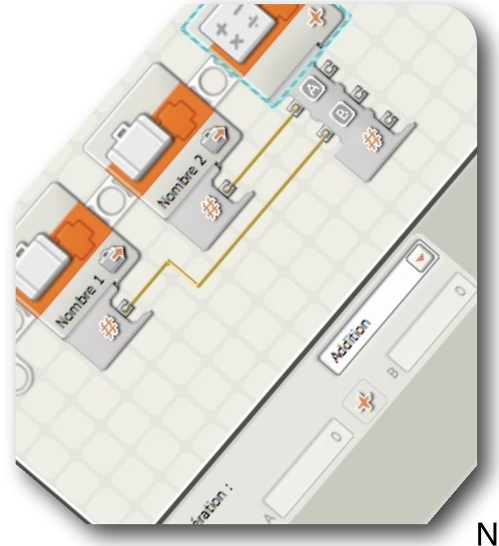
Fig.16



Exécutez à présent le programme, à plusieurs reprises en enfonçant et lâchant le bouton ENTREE, ou ne faîtes rien. Votre choix d'appuyer ou non le bouton, convertira en valeur VRAI ou FAUX ce qui est rangé dans le bloc variable ENFONCE. Après 3 secondes, la dernière valeur ainsi rangée sera lue par le bloc COMMUTATEUR, et le texte approprié affiché sur l'écran.

Quel intérêt avons nous de procéder ainsi? On aurait pu tout simplement relier le bloc BOUTON NXT au bloc COMMUTATEUR…
En effet, si vous n'envisagez pas de changer d'avis.
Mais si vous souhaitez inverser votre décision, vous avez 3 secondes pour le faire; et dans ce cas, c'est possible sans être obligé de modifier votre programme.
La variable ENFONCE a tout simplement conservé en mémoire la dernière valeur logique, valeur qui a été utilisée par la suite. Cette valeur peut être réutilisée à n'importe quelle moment jusqu'à la fin du programme.

Retenez simplement que les variables constituent un moyen puissant de stockage d'informations que votre robot peut utiliser plus tard.



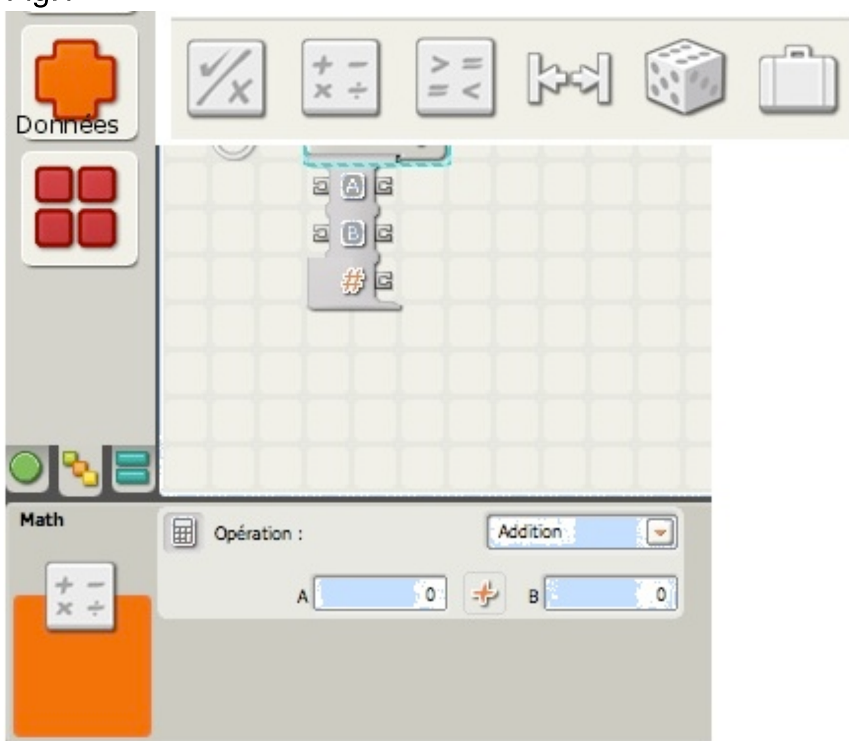


Leçon n°10: OPERATION...CALCULS

Ceux qui aiment les maths seront servis. Vous verrez comment avec quelques blocs il est possible d'effectuer n'importe quelle opération.

N'oubliez pas que le NXT est d'abord un calculateur, aussi sachez vous en servir dans les programmes. Nous allons d'ailleurs les examiner; d'abord pour des opérations simples, puis en combinaison avec d'autres blocs faisant intervenir quelques notions particulières.
Voici à quoi ressemble ces blocs. Vous les trouverez dans la palette *Complète*, en survolant l'icône *Données*

Fig.1

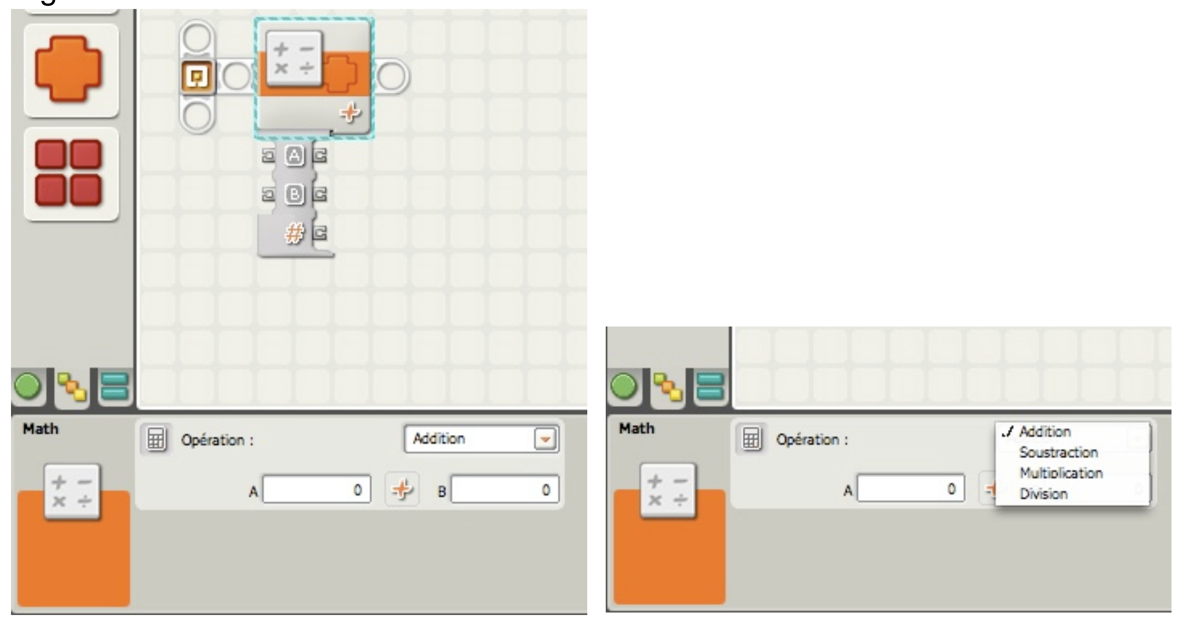


Calculs basiques:

Rappelons d'abord une règle fondamentale:
*** 3ème règle: Un bloc ne peut accomplir qu'une seule tâche à la fois. Il faut autant de blocs que de tâches à accomplir.**
Cela signifie un bloc pour chaque opération.

addition, soustraction, multiplication et division

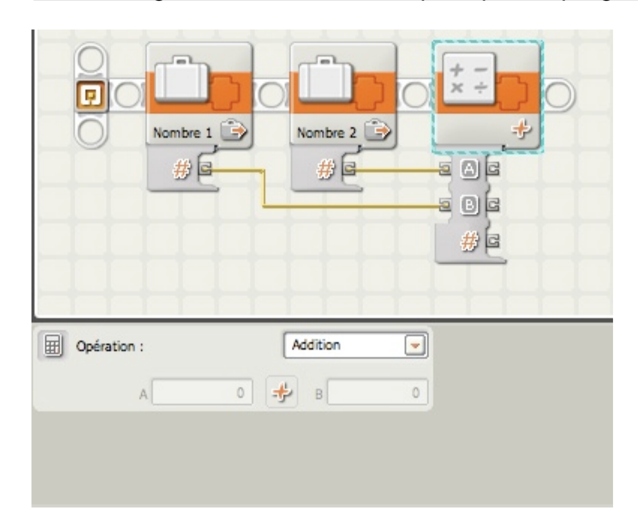
Fig.2 Bloc MATH



Le type d'opération est sélectionné à partir du champ de saisie. Ils se comportent tous de la même façon.
Ce bloc emploie 2 valeurs *numériques* A et B. Ces valeurs peuvent être positives, négatives et seulement en entier. <u>Cela signifie que les valeurs décimales ne sont pas acceptées</u>. Si vous tentez d'écrire une valeur comme -5,3 ou 8,7, le bloc MATH arrondira cette valeur à la valeur entière la plus proche soit -4 et 9 pour notre exemple.
Le menu déroulant permettant de choisir l'opération donne ceci:
Addition: A et B sont additionnés.
Soustraction: B est soustrait à A
Multiplication: A est multiplié par B
Division: A est divisé par B.
Vous remarquerez que les plots de données acceptent deux entrée pour A et B, et trois sortie pour A, B et le résultat de l'opération, qui est une valeur numérique. Rien d'autre; ce bloc est très simple et facile à utiliser.

Si aucune valeur est saisie dans les champs A et B, le zéro (0) est la valeur de défaut. Ces nombres peuvent être saisie à la main ou fournis par un autre bloc, à l'aide d'un fil de données raccordé à cet autre bloc. En voici un exemple:

Fig.3

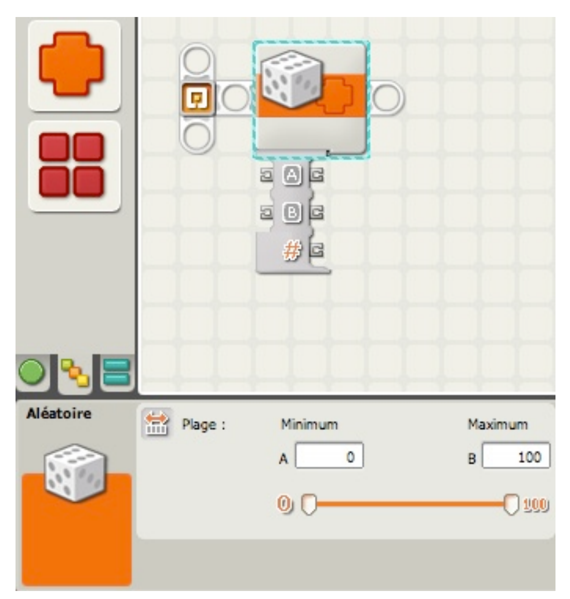


Vous devez comprendre que vous ne pouvez rien faire avec le bloc MATH si les plots de données ne sont pas accessibles. La raison est simple; quel que soit le type d'opération à effectuer, le résultat se trouve dans le plot de sortie, et si vous voulez l'utiliser, vous avez besoin d'un fil de données à raccorder à un bloc qui vous permettra de lire le résultat sur un écran d'affichage par exemple.

Calculs aléatoires

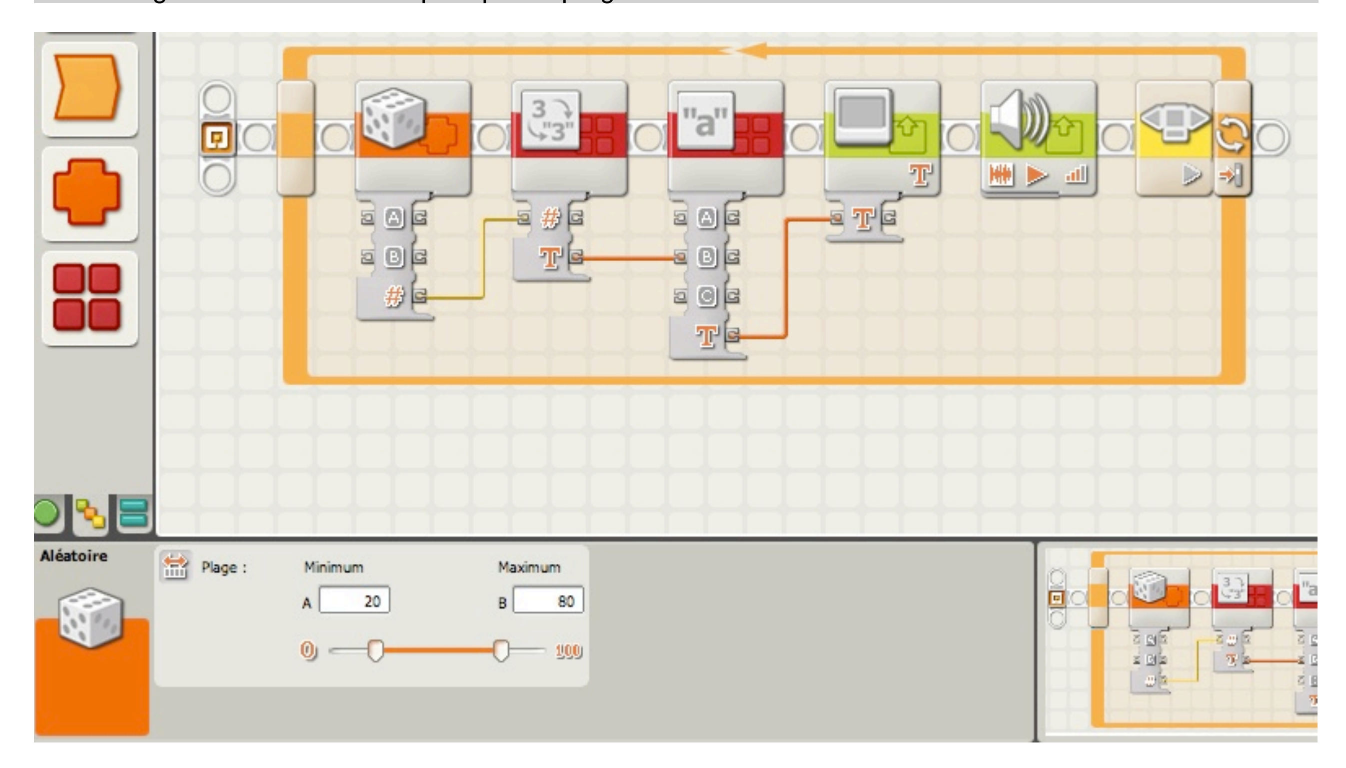
Imaginons un jeu de 52 cartes battues et étalées, faces cachées sur une table, et que vous en choisissiez une au hasard. Vous avez 1 chance sur 52 de tirer celle que vous avez en main. Si vous recommencez la même opération, vous avez toujours 1 chance sur 52 de tirer n'importe quelle autre carte, y compris celle déjà tirée (puisqu'elle n'a pas été retirée du jeu).
Voyons autrement à l'aide d'une paire de dés à jouer. Chaque dé en roulant, a la faculté de monter sur chacune de ses faces une valeur de 1 à 6. Quand vous lancez une paire de dés sur un tapis, les valeurs qui apparaissent sont quelconques. Les chances d'avoir un 3 sur un dé sont les mêmes que celles d'avoir un 6. Elles ont toutes la même probabilité.
Aléatoire: c'est ainsi que l'on désigne ce phénomène de probabilité. Il ne suit aucune règle, aucune logique; il est imprévisible.
Votre robot est lui aussi capable de générer des nombres aléatoires. Vous pouvez construire un robot susceptible de lancer une paire de dés virtuels, ou de tirer au hasard un nombre entre 1 et 1000. C'est ce que nous allons voir maintenant.

Bloc ALEATOIRE
C'est un bloc d'usage très simple, avec peu de paramètres de configuration, mais importants.
Fig.4



Dans la zone *Plage*, vous remarquerez la présence de 2 champs de saisie: Minimum (A) et Maximum (B). Dans ces champs vous préciserez les valeurs haute et basse de la plage dans laquelle le bloc "piochera" d'une manière aléatoire. La valeur minimum peut être inférieure à zéro (0), et la valeur maximum ne peut dépasser 32 737.
Une autre façon de choisir ces limites minimale et maximale est de tirer les deux glissières. La limite supérieure de la glissière est fixée à 100 ; toutefois, si vous entrez une valeur dans la zone de saisie, vous pouvez fixer la limite supérieure au-delà de 100. Vous pouvez également entrer un nombre négatif dans la zone de la limite minimale ; ce nombre supplantera la valeur de la glissière.
Pour fixer les limites minimale et maximale de manière dynamique, branchez des fils de données en entrée au plot de données du bloc.

Exemple:
Nous allons maintenant écrire un *pseudo-code*:
LEO > BONG, affiche sur l'écran du NXT et signale par un top sonore, un nombre aléatoire compris entre 20 et 80. Recommence jusqu'à ce que j'appuie sur le bouton flèche droite pour tout arrêter.
Et voilà à quoi ressemble le programme NXT-G.
Fig.5



Le bloc NOMBRE EN TEXTE permet l'affichage de la valeur aléatoire en passant par le bloc TEXTE (qui lui contient une description en A et la valeur convertie en B). Un top sonore ponctue l'affichage du résultat. La boucle recommence la même opération avec une autre valeur aléatoire, jusqu'à ce que la flèche droite soit appuyée. On termine alors le programme.
Autre application: si vous souhaitez faire évoluer votre robot d'une manière indépendante de votre volonté, vous programmerez en utilisant ce bloc qui fournira justement une valeur fruit du hasard, qui conditionnera ses mouvements (voir leçon n° 8).

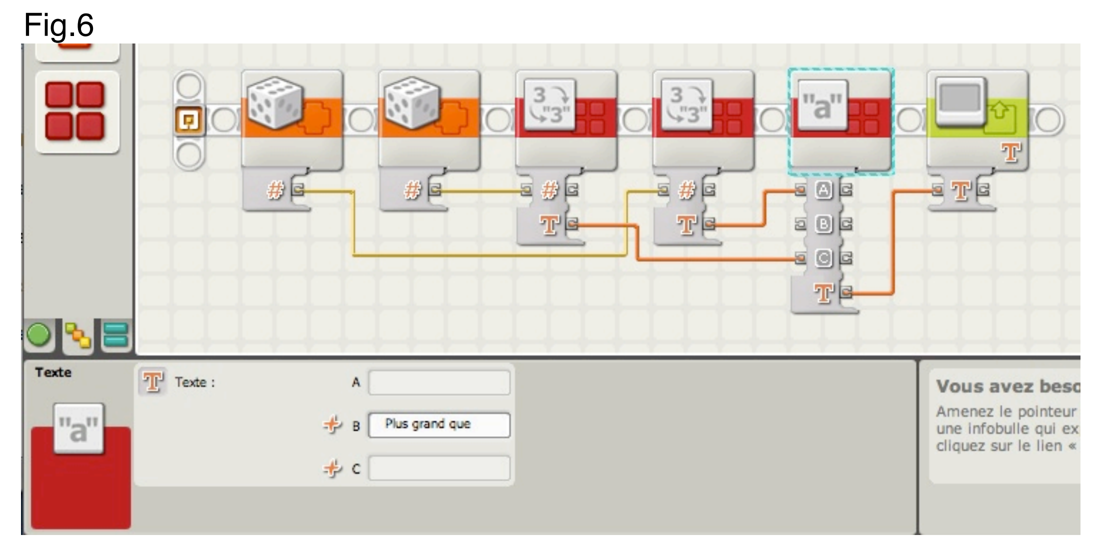
Comparaison

Vous connaissez le vieil adage: "on ne mélange pas les torchons avec les serviettes!". Mais parfois cela pourrait être utile de comparer des objets dissemblables. Comme par exemple: qui est plus lourd, qui est plus grand?
Votre robot peut aussi faire de même; non pas comparer des objets, mais des nombres. Et cela grâce à une particularité des bloc NXT-G qui manipules des valeurs Logiques.
Supposons que je vous dise: "5 est-il plus grand que 3?". Votre réponse est sans aucun doute OUI. Lorsque vous programmez votre robot de la même manière (5 est plus grand que 3), la réponse ne peut-être que VRAI ou FAUX; VRAI dans notre cas.
De même, si je vous demande: "la lune est-elle de forme cubique?", votre réponse serait NON et votre robot répondrait FAUX à la même question.
Aussi, n'oubliez pas qu'un robot répond à des ordres (des instructions) par VRAI ou par FAUX.

Avant de présenter le bloc COMPARER, nous allons écrire un *pseudo-code* pour tester les connaissances de BONG.
LEO > BONG, je veux que tu crées 2 nombres aléatoires compris entre 1 et 9 (nombre A et nombre B); affiche-les sur l'écran, et dis moi si A est plus grand que B.
Je vais donc commencer par glisser 2 blocs ALEATOIRE et les configurer d'une manière identique (A=1 et B=9).
Après avoir convertis les valeurs en mode texte (à l'aide de 2 blocs NOMBRE EN TEXTE identiques), je les regroupe dans un bloc TEXTE unique. Enfin, je les affiche sur l'écran du

NXT (ligne 3, x=2 et y=40). J'aurai donc une expression sous la forme; ***(valeur A) Plus grand que (valeur B).***

Fig.6



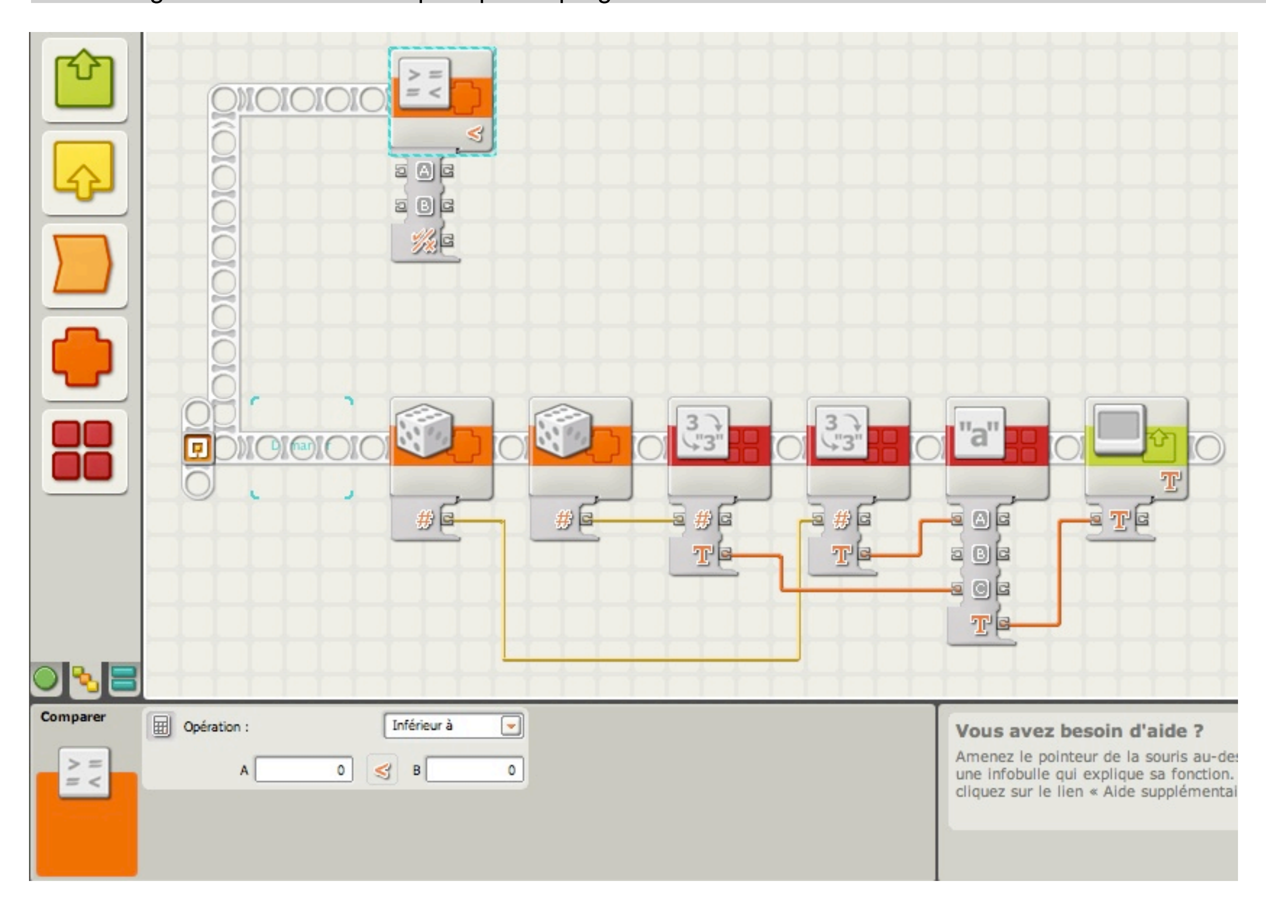
Remarque: dans le champ B, taper ">" précédé d'un espace (le texte de la fig.6 est trop long; il "mange" la suite.

Nous allons maintenant faire dire par BONG si l'expression est VRAI ou FAUX, et nous sommes prêt pour cela à utiliser le bloc COMPARER.
Désolé d'avoir été si long pour expliquer ce bloc (qui par lui-même est incapable de faire quoi que ce soit), mais un bon exemple en situation est nécessaire pour comprendre le mécanisme.

Bloc COMPARER

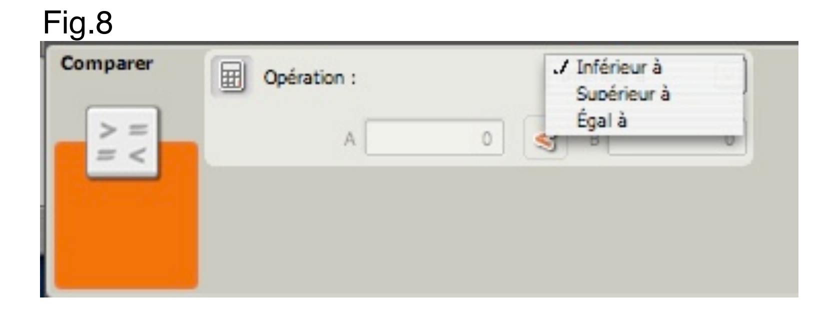
Je vais à présent créer un autre RAYON DE SEQUENCE en parallèle avec ce dernier. Je rappelle à cet effet que vous pouvez utiliser le point de départ pour créer des rayons de séquence supplémentaires qui permettent à votre programme d'accomplir des tâches simultanées (pour créer un rayon supplémentaire voir le menu AIDE du logiciel NXT-G). Sur ce rayon je glisse un bloc COMPARER (pour comparer les valeurs A et B). Rappelez-vous: je veux vérifier l'expression **A Plus grand que B** et m'assurer que le résultat est **VRAI** ou **FAUX**.

Fig.7



Observons le panneau de configuration.

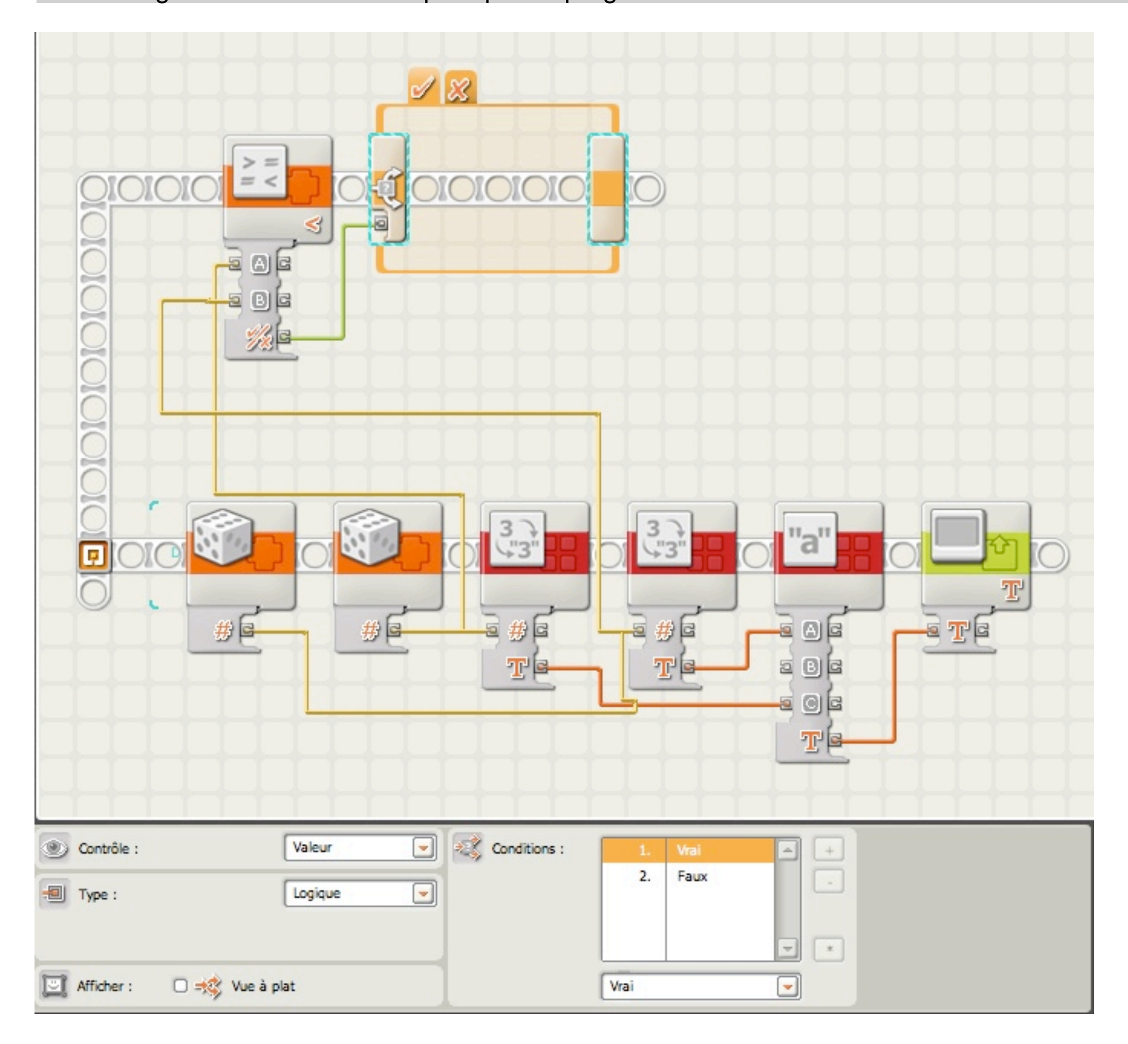
Fig.8



Ce bloc dispose d'un menu déroulant dans la zone Opération. Trois options sont visibles. Si vous choisissez l'option "Inférieur à", le bloc COMPARER déterminera si **A est Inférieur à B** et traduira l'expression par **VRAI** ou **FAUX**. Il procédera de même si vous choisissez une autre option.
J'ai choisi cette option, et le bloc fournira la réponse par le plot de sortie du résultat.

Vous remarquerez aussi que le bloc COMPARER dispose de 2 plots d'entrée. Je vais raccorder par un fil de données chacun de ces 2 plots a un plot de chaque bloc NOMBRE EN TEXTE.
Si nous revenons maintenant au *pseudo-code,* BONG doit m'afficher le mot VRAI ou FAUX (selon les valeurs fournies par le bloc ALEATOIRE) et vérifier l'expression "A plus grand que B". Pour accomplir cette tâche, je vais ajouter un bloc COMMUTATEUR.

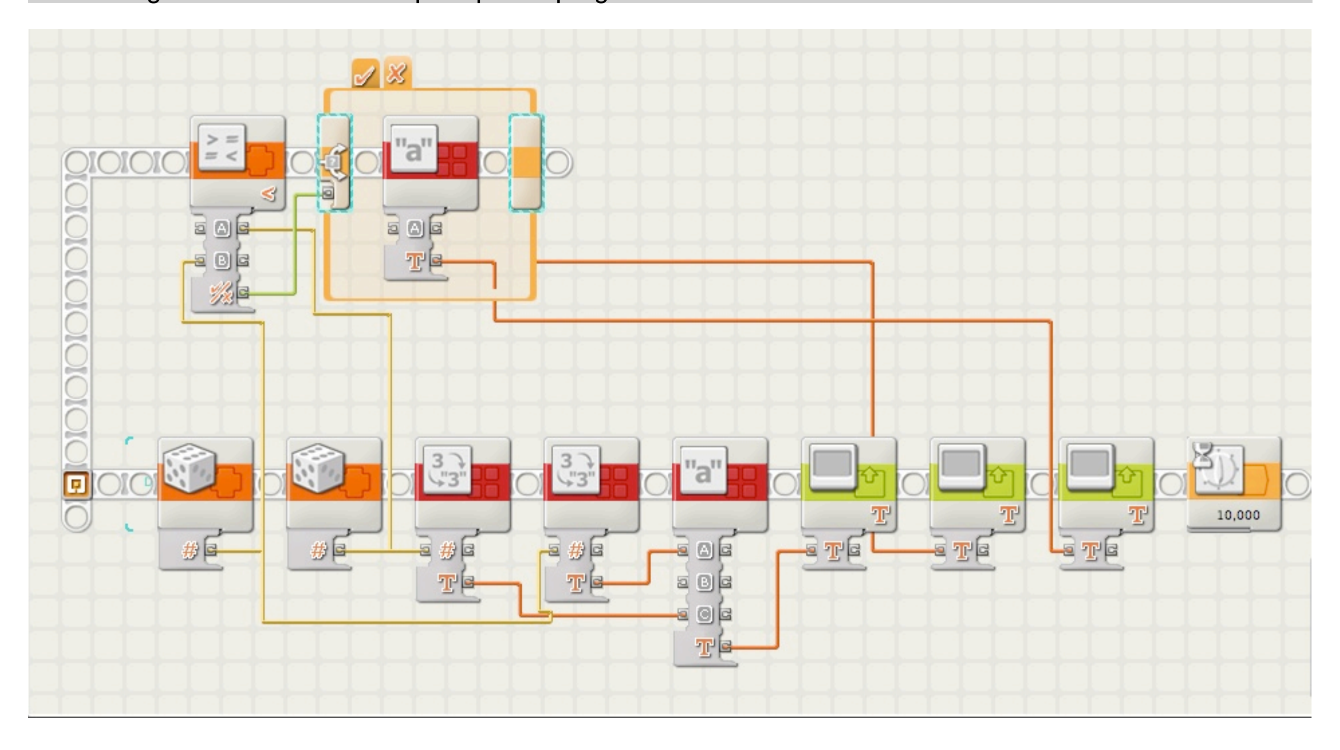
Fig.9



Dans le panneau de configuration, j'ai choisi "Valeur" et "Logique", et décoché "Vue à plat". J'ai de plus tiré un fil de données entre le plot de sortie du bloc COMPARER et le plot d'entrée du bloc COMMUTATEUR.

Je dois maintenant décider de ce qu'il adviendra lorsque le bloc COMMUTATEUR recevra un signal VRAI ou FAUX du bloc COMPARER.
Je veux lire le mot VRAI , si l'expression est vérifiée. Je vais donc placer un bloc TEXTE contenant le mot "VRAI".
Mais, pour rendre ce mot visible sur l'écran, je vais ajouter un autre bloc AFFICHER (en décochant "Effacer") en bout du rayon de séquence principal, puis tirer un fil de données vers le bloc TEXTE.
Pour compléter l'exemple, je procéderai de la *même manière* dans le cas où l'expression n'est pas vérifiée:
Dans le bloc COMMUTATEUR je sélectionne l'onglet X , je positionne un nouveau bloc TEXTE contenant le mot "FAUX"; j'ajoute un autre bloc AFFICHER sur le rayon de séquence principal relié à son tour par un fil de données au dernier bloc TEXTE (FAUX).
Et pour terminer, Un bloc ATTENDRE pour afficher suffisamment longtemps pour la lecture.

Fig.10



Je vous conseille d'écrire vous-même et de tester ce programme. Il est recommandé pour une meilleure compréhension et une bonne pratique des fils de données. Modifiez-le à votre guise; changez également les conditions pour "Supérieur à" ou "Egal à", et observez comment les résultats se modifient.

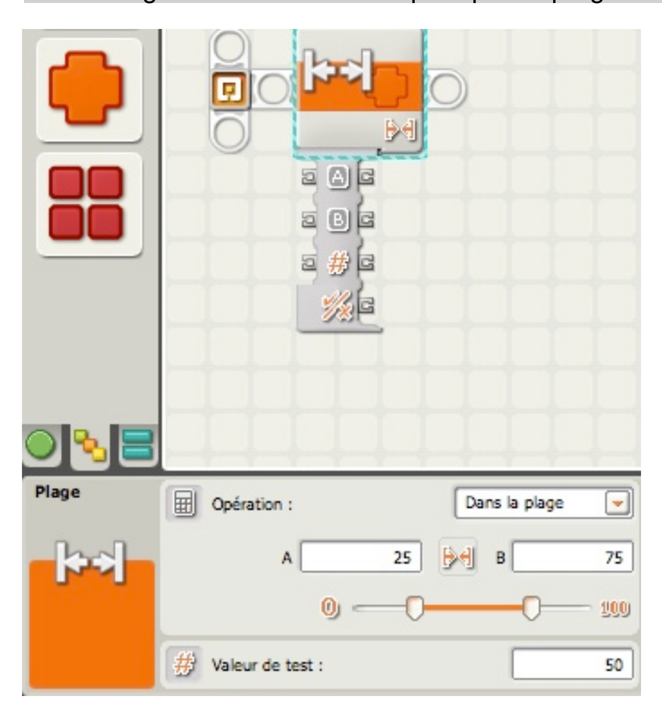
Plage

Il arrive parfois que la comparaison entre deux valeurs ne suffit pas pour résoudre une notion de logique. Où la question est de vérifier si une valeur se trouve à l'intérieur ou à l'extérieur d'une plage de nombres.
Est-ce que par exemple, 17 est-il dans la plage 5 - 22? Ou bien, 40 est-il en dehors de la plage 1 - 10?
Quelle utilité, demanderez-vous, peut-il y avoir ?
Et bien, dans le cas du bloc COMPARER seules 2 valeurs sont en cause: A et B.
dans le cas qui nous préoccupe, nous affinons l'approche logique en y introduisant 3 valeurs: A, B et C. Nous testons ici la valeur A par rapport à une série d'autres valeurs comprises entre B et C, pour en déduire une relation logique VRAI ou FAUX. Le NXT-G dispose à cet effet d'un bloc particulier que nous allons maintenant aborder.

Bloc PLAGE

Tout comme le bloc COMPARER, ce bloc répond à la règle suivante:
le résultat obtenu se présente sous la forme Logique VRAI ou FAUX.

Fig.11



Le bloc PLAGE possède un plot de données avec trois ports d'entrée (à gauche) et quatre ports de sortie (à droite). Les ports d'entrée peuvent être connectés à d'autres blocs à l'aide de fils de données (sauf si un ou plusieurs des ports d'entrée reçoit un nombre que vous entrez).

La sortie *Logique* du bloc sera transmise à partir de la prise de sortie inférieure, à laquelle vous devez connecter un fil de données relié au plot de données d'un autre bloc. Les trois prises de sortie en face des prises d'entrée permettent, si nécessaire, de transmettre les nombres en entrée à d'autres blocs.

La valeur de test peut être tapé ou fourni de manière dynamique par un fil de données. La zone de test est grisée si un fil de données est connecté.
Les limites inférieure et supérieure peuvent être définies à l'aide de la glissière, être entrées ou être fournies de manière dynamique par des fils de données. Les zones de saisie sont affichées en grisé lorsque des fils de données sont connectés.
Le menu déroulant permet de choisir l'une des deux opérations disponibles :

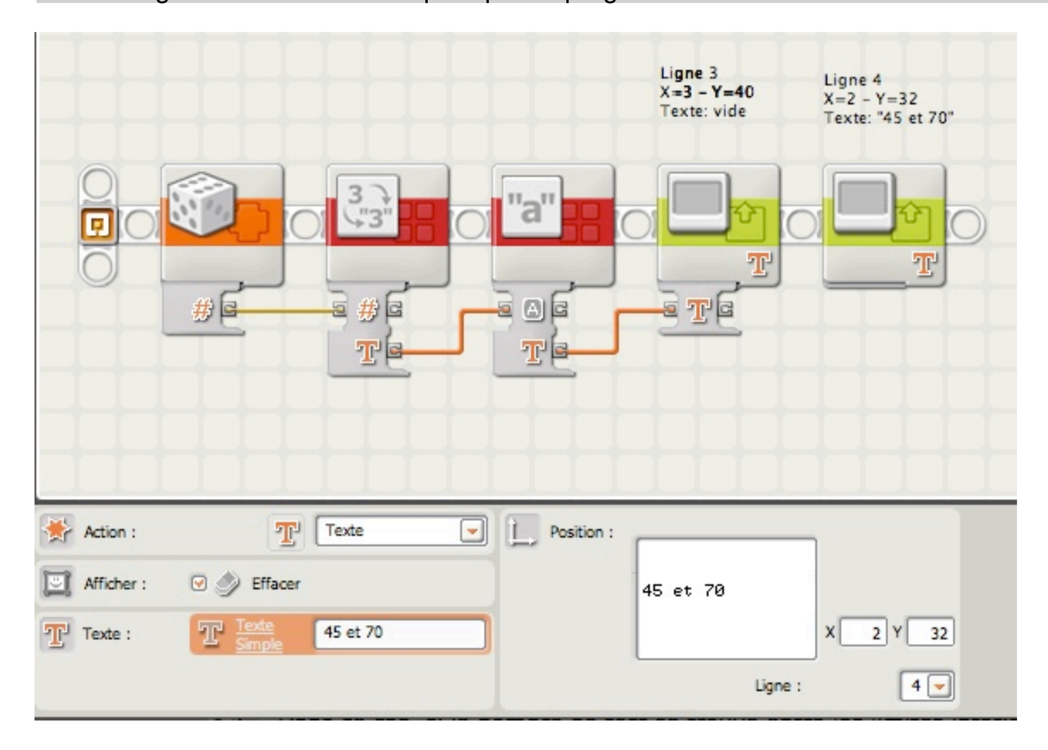
* Dans la plage ([ ])
* Hors de la plage (] [)

Ce bloc vérifiera les instructions suivantes:
A est à l'intérieur de la plage qui commence par B et se termine par C.
A est en dehors de la plage qui commence par B et se termine par C.
La réponse sera VRAI ou FAUX en fonction de ces vérifications.

Un petit exemple en *pseudo-code* pour illustrer son utilisation.

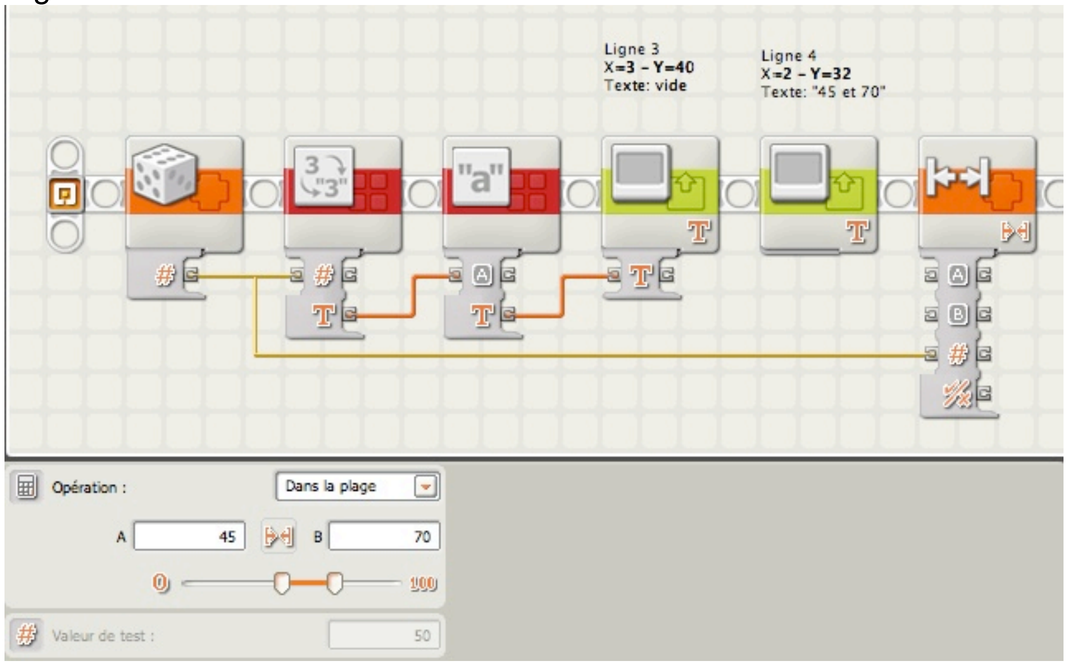
LEO > BONG, Génères un nombre aléatoire compris entre 1 et 100; affiches cette valeur sur ton écran, et dis moi moi si cette valeur est comprise entre 45 et 70.
Commencez par écrire ce début de programme:

Fig.12



Ajoutez à présent le bloc PLAGE configuré ainsi:

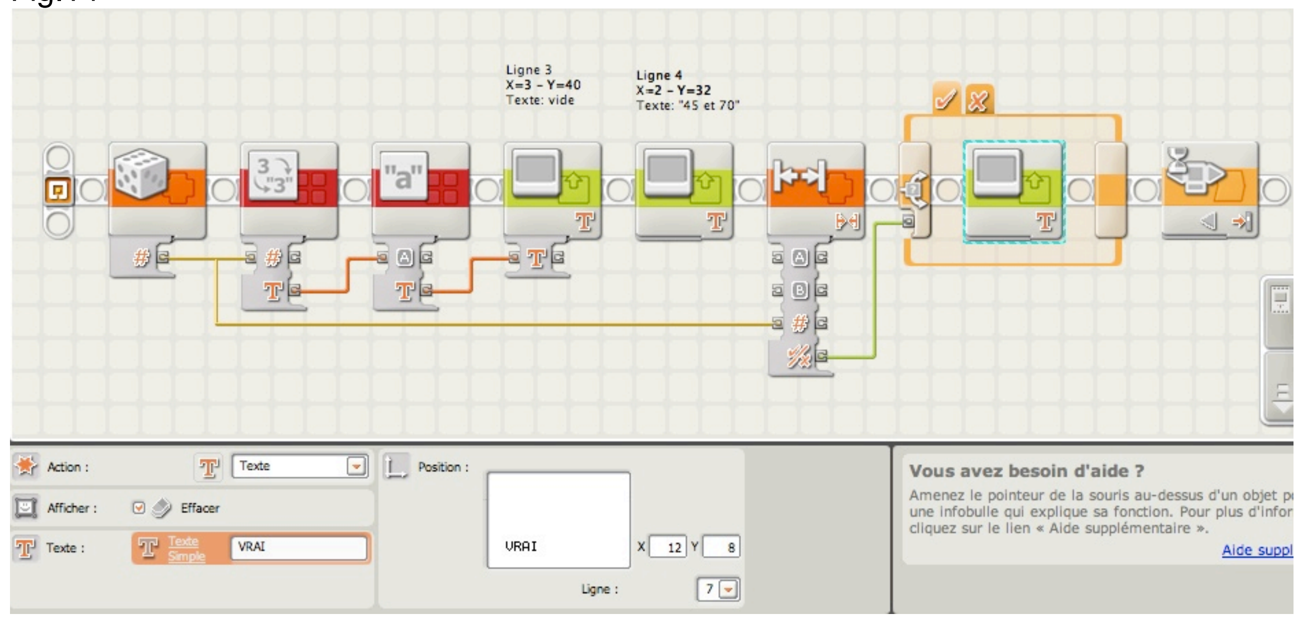
Fig.13



Observez le panneau de configuration définissant la plage 45 - 70 ainsi que le fil de données liant le bloc ALEATOIRE au bloc PLAGE.

Il ne reste plus qu'à ajouter un bloc COMMUTATEUR qui contiendra un bloc AFFICHER dans chacun des chemins, pour visualiser par les mots VRAI ou FAUX le résultat du tirage.
Et pour finir, un bloc ATTENDRE avec un bouton NXT flèche Gauche qui permet de maintenir l'affichage avant de quitter le programme. (Ne vous trompez pas de bloc: il s'agit d'un bloc ATTENDRE, et non pas un bloc Bouton NXT)).

Fig.14



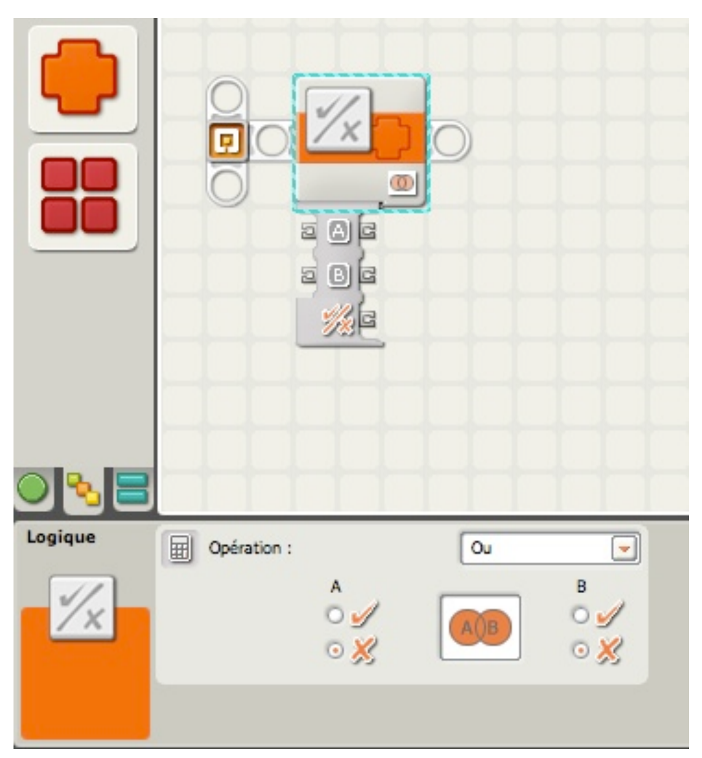
Exercez-vous avec ce programme, essayez l'option "Hors de la plage" et changez les limites de la plage. Poursuivez cet exercice jusqu'à obtenir une bonne maîtrise de l'usage de ce bloc.

Logique

Bloc LOGIQUE

Ce bloc est très proche du bloc COMPARER avec une différence fondamentale dans la natures des valeurs .
Alors que le bloc COMPARER manipule des valeurs numériques, celui-ci compare des valeurs LOGIQUES et donne comme réponse une valeur logique VRAI ou FAUX.
Avez-vous noté la différence? En effet il est possible de comparer des valeurs logiques; on entre ici dans la théorie des Ensembles, et on aborde un domaine d'opérations particulières.
C'est le bloc qui a ma préférence, celui de "l'élégance", celui qui par raisonnement logique permet l'écriture concise d'un programme. C'est celui qui économise de longues chaînes de blocs, celui de l'allégement.

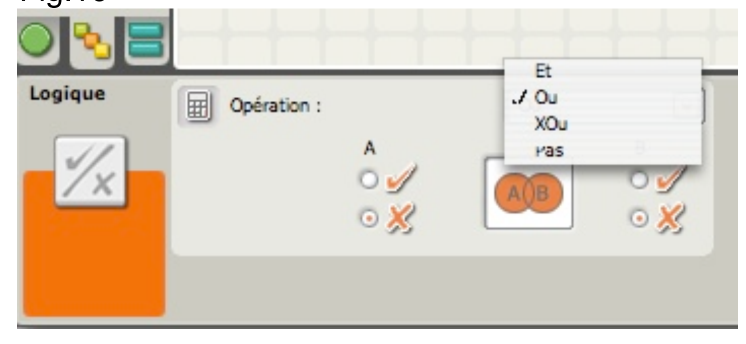
Fig.15



Examinons le panneau de configuration:

1. Utilisez les cases d'option pour choisir les entrées A et B, ou laissez des fils de données en entrée fournir les valeurs d'entrée de manière dynamique.
2. Le menu déroulant permet de choisir l'une des quatre opérations que vous pouvez appliquer aux ***entrées*** :
   * Opération « Et »
   * Opération « Ou »
   * Opération « Ou exclusif »
   * Opération « Non »

Fig.16



Ces opérations font référence à l'algèbre de Boole dite algèbre combinatoire.
Le tableau ci après résume à l'aide d'icônes représentatives, les résultats qui en découlent.

Fig.17

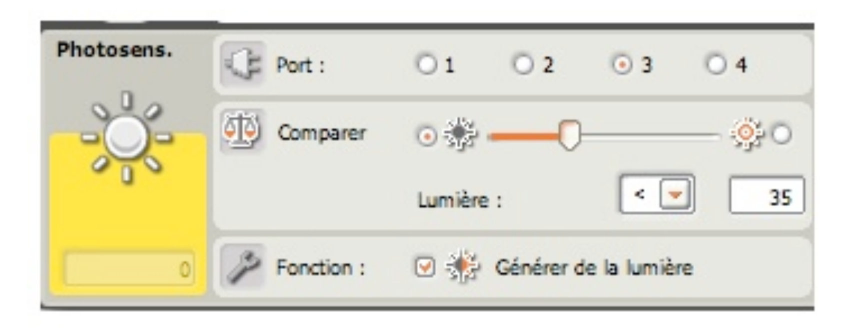
Opération	Représentation	Caractéristiques
Et		si les deux valeurs en entrée sont toutes deux vraies, la sortie vaut elle aussi « vrai ». Dans tous les autres cas, la sortie vaut « faux ».
Ou		si l'une des valeurs en entrée, ou les deux, sont vraies, la sortie vaut « vrai ».
Xou Opération dite « Ou exclusif »		si l'une des valeurs en entrée est vraie, mais pas les deux, la sortie vaut « vrai ».
Pas L'opération « Non » est parfois baptisée « inversion ».		Cette opération n'emploie qu'une valeur en entrée. Si cette valeur est vraie, la sortie vaut « faux » ; si la valeur est fausse, la sortie vaut « vrai ». Cette opération se contente donc d'inverser la valeur en entrée.

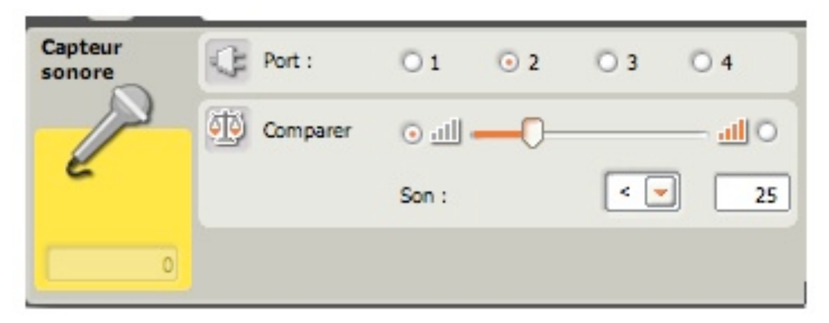
Nous allons créer un petit scénario pour BONG, et j'espère que cela vous aidera à comprendre comment ce bloc fonctionne.
J'ai fixé et branché sur BONG un capteur Photosensible et un capteur Sonore. Le *pseudo-code* que j'imagine est le suivant:

LEO > BONG, avance de 3 rotations (du moteur), si ces 2 conditions sont vérifiées (VRAI).
LEO > La 1ère condition est que le capteur Photosensible détecte un niveau lumineux inférieur à 35.
LEO > la 2ème condition est que le capteur Sonore détecte un niveau sonore inférieur à 25.

Bien, que va-t-il se passer? BONG vérifiera si le capteur lumineux détecte un faible niveau dans le local (< 35). Il vérifiera également si le capteur sonore détecte un niveau calme (< 25). Il enverra une valeur VRAI ou FAUX selon les conditions de configuration qui ont été paramétrées.
Le programme commencera par l'installation des 2 blocs capteurs sur le rayon de séquence avec leurs panneaux de configuration:

Fig.18

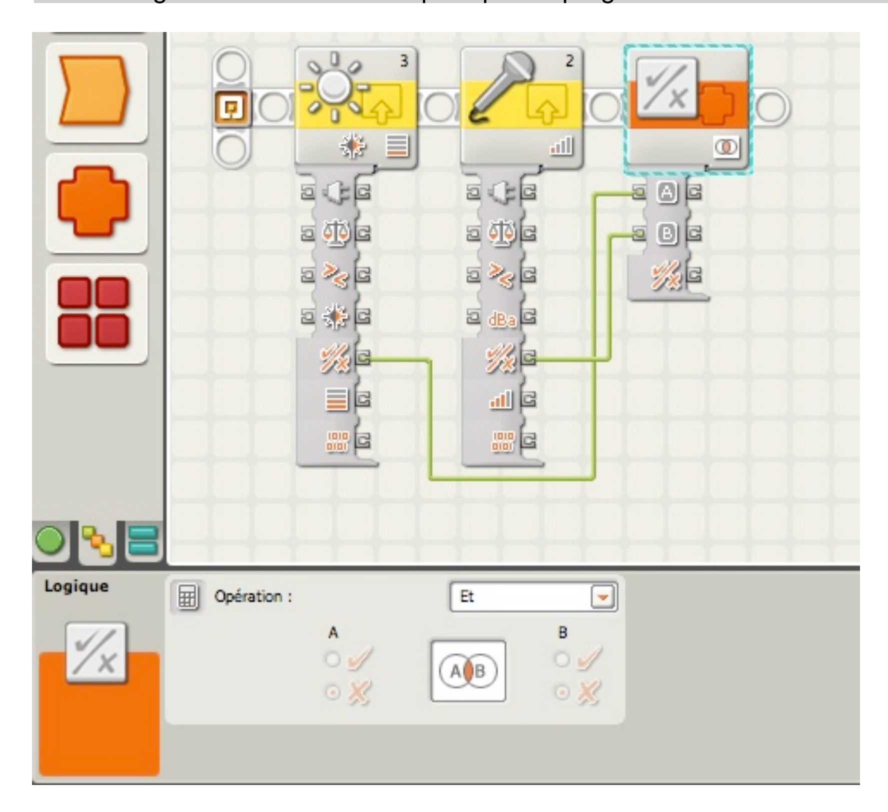




Si on revient au *pseudo-code*, BONG avancera dans le cas unique où les 2 capteurs franchiront simultanément leurs points de déclenchement. Cela signifie que le capteur Photosensible doit réagir à une valeur inférieure à 35, pour le niveau lumineux du local, ET le capteur Sonore à une valeur inférieure à 25 pour le niveau sonore.
Les 2 conditions doivent être vérifiées (VRAI) en même temps, sinon BONG ne bougera pas.
Que se passe-t-il si le local est très lumineux et l'ambiance calme? RIEN; BONG reste immobile.
Que se passe-t-il si le local est dans l'obscurité et l'ambiance bruyante? RIEN également; BONG reste toujours immobile.
Que se passe-t-il si le local est lumineux et bruyant? Un gros mal de crâne pour BONG, mais il ne bougera toujours pas.

Comment BONG peut-il vérifier rapidement les conditions de luminosité et de bruit du local, et décider d'avancer ou pas?
Simple, il utilisera un bloc LOGIQUE, et le programme ressemblera à ceci:

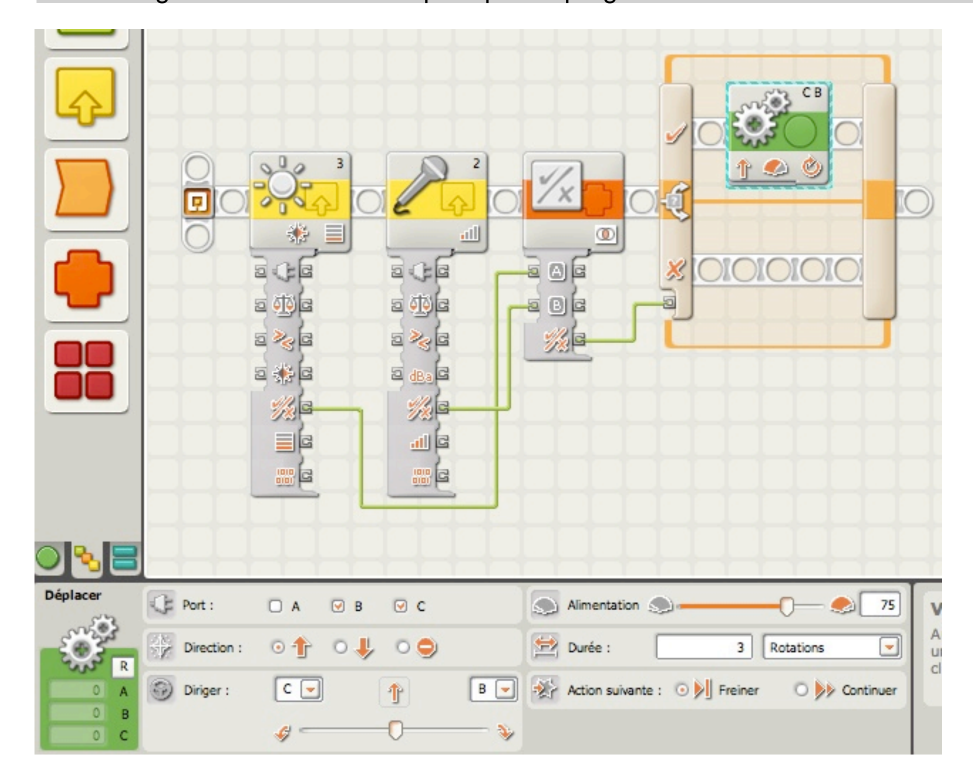
Fig.19



Le bloc LOGIQUE fonctionne comme un bloc COMPARER avec une particularité: c'est celle de pouvoir combiner à l'aide d'opérations (d'un type nouveau), des valeurs Logiques transmises par les capteurs, et de fournir un résultat qui est lui aussi une valeur Logique. Ces opérations ont été indiquées dans le tableau ci dessus, et dans notre cas nous avons choisi **Et** dans la Zone correspondante.
Pour éviter toute erreur, l'icône visualise le type de résultat recherché. On remarque ici la partie colorée en rouge de la zone commune des 2 cercles. Cette zone est bien commune aux 2 valeurs et correspond à l'objectif recherché.

Nous compléterons ce programme par un bloc COMMUTATEUR qui orientera vers le chemin répondant à ces 2 conditions, chemin contenant un bloc AVANCER.

Fig.20



Ce bloc est un magnifique outil qui combine avec art 2 valeurs Logiques. Il permet de manipuler un grand nombre de valeurs, dont les résultats peuvent-être associés à d'autres valeurs logiques.
Pour combiner, par exemple 4 valeurs logiques, j'utiliserai 2 blocs LOGIQUE et le résultat de chacun sera envoyé sur un 3ème bloc LOGIQUE qui fournira le résultat final.

Compliqué? Un peu en effet. Mais quand vous utiliserez ce bloc, commencez par l'associer aux BOUCLES et aux blocs COMMUTATEURS pour donner au robot de meilleures et plus larges possibilités de décision.



Nous abordons ici une partie des plus intéressantes de la programmation NXT-G, celle qui vous permet d'élargir considérablement vos palettes de blocs.

Parmi toutes les palettes à notre disposition, il en est une que nous n'avons pas encore examiné: pour cela, vous allez choisir la *palette personnalisée*, puis survoler l'icône "*Mes Blocs*".

Vous ne voyez pas grand chose, n'est-ce pas? En effet, "Mes Blocs" ne contient aucun bloc. Et pour cause, parce que ce sont les vôtres, ceux que vous n'avez pas encore créés, mais que vous allez créer!

Et pourquoi créer des blocs dans "Mes Blocs"?
Parce que vous avez constaté en écrivant vos programmes NXT-G que certaines parties pouvaient être réutilisées telles quelles dans des programmes différents. Et que les réécrire à chaque fois, c'était souvent du "copier-coller" (pour éviter aussi de commettre des erreurs). Alors pourquoi ne pas utiliser un moyen de simplification et de gain de temps?
Et que pourrait-on trouver dans ces morceaux de programme? Par exemple le contrôle des roues, des messages sonores, des calculs intermédiaires, etc..
Tous ces morceaux constituent une collection de Blocs NXT-G qui seront groupés dans "Mes Blocs" pour être réutilisés à volonté dans tous les programmes que vous écrirez. C'est ce que je vais essayer de vous expliquer…

Pour illustrer le propos, nous allons reprendre le programme NXT-G figurant en tête de la leçon n°7 avec le *pseudo-code* qui l'accompagne:


Figure 1



LEO > BONG, avance de 3 rotations (du moteur), arrête-toi et tourne à droite de 90 degrés.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

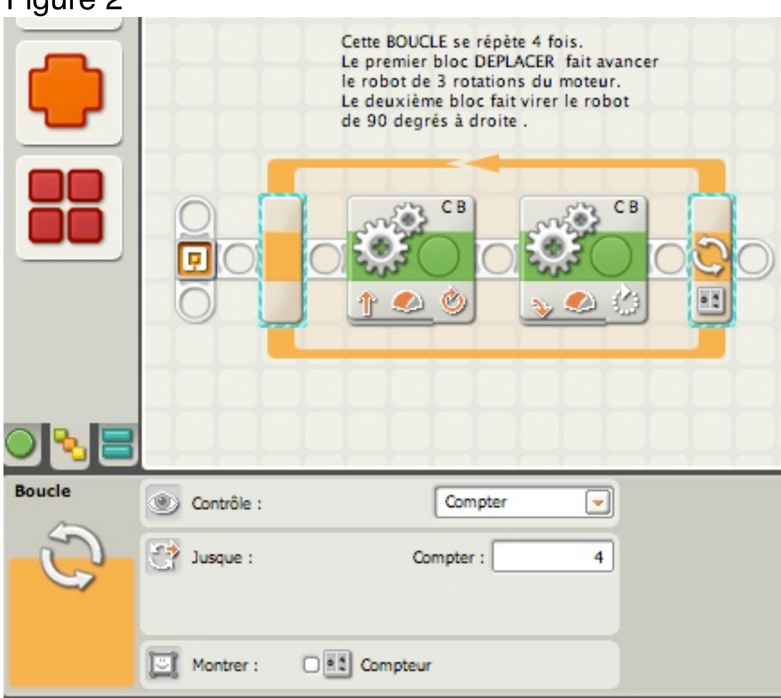
LEO > BONG, avance de 3 rotations (du moteur), arrête-toi et tourne à droite de 90 degrés.
(BONG avance à nouveau de 3 rotations, s'arrête puis tourne à droite…)

LEO > BONG, avance à nouveau de 3 rotations (du moteur), arrête-toi et tourne à droite de 90 degrés.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

LEO > BONG, avance à nouveau de 3 rotations (du moteur), arrête-toi et tourne à droite de 90 degrés.
(BONG avance de 3 rotations, s'arrête puis tourne à droite…)

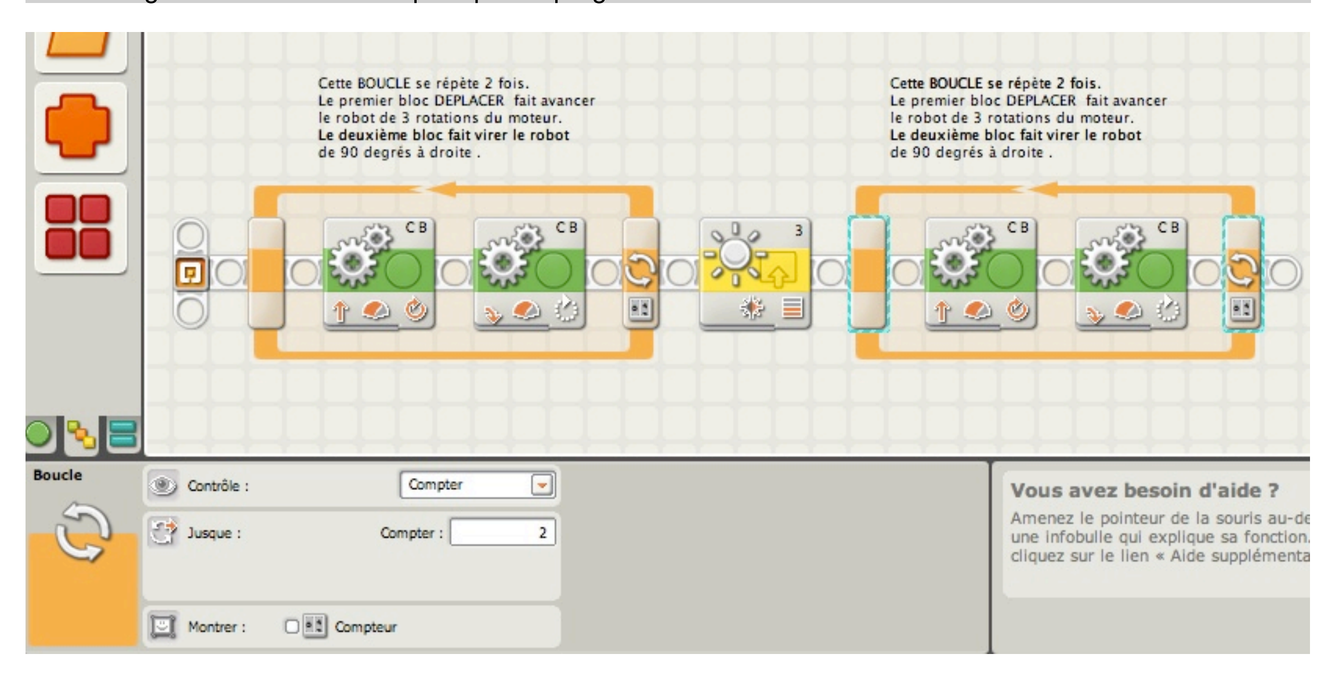
Je peux aussi simplifier cette suite de 8 blocs et la remplacer par une simple BOUCLE, solution plus élégante.

Figure 2



Bien. Mais que se passe-t-il si j'envisageais faire réagir BONG au capteur PHOTOSENSIBLE après 2 tours de boucle?
Il faudrait donc que je modifie le programme de la figure 1 en insérant au milieu un bloc Capteur PHOTOSENSIBLE, ce qui ne pose aucun problème.
Mais pour le programme de la figure 2, je dois modifier la BOUCLE en remplaçant *Compter* 4 par 2. Puis, ajouter un bloc Capteur PHOTOSENSIBLE suivi d'une 2ème BOUCLE identique à la première.
Le programme ressemblerait à ceci:

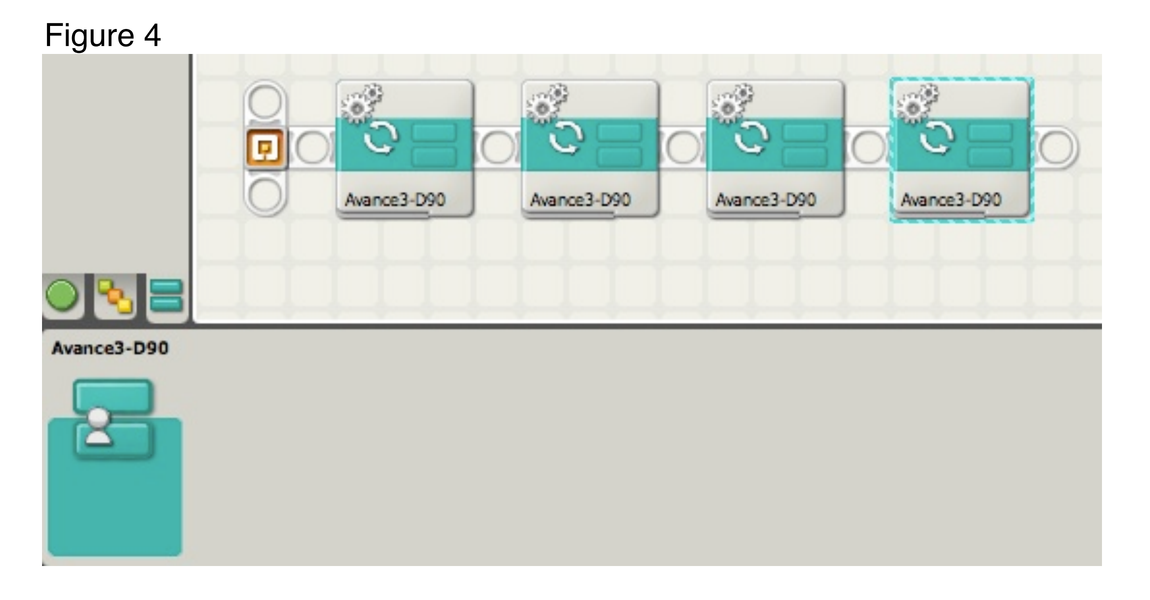
Figure 3



Mon BLOC

Modifier ce programme était relativement facile, mais imaginez que le programme devienne de plus en plus long avec d'autres contraintes, comme tourner à droite ou à gauche. Les ajouts successifs de blocs DEPLACER peuvent l'alourdir considérablement. C'est là que le concept de *Mes Blocs* entre en jeu.
Commencez par examiner ce programme; je le commenterai par la suite.

Figure 4



Vous voyez ici 4 blocs *Mon Bloc* dont les icônes ne figurent pas dans les palettes courantes. Chacun d'entre eux contient les 2 premiers blocs DEPLACER de la figure 1. Tout ce que j'ai fait consiste à les regrouper en un seul Bloc. J'ai de la sorte réduit la longueur du programme de moitié.
Ces blocs ont un nom: *Avance3-D90*, et une *icône personnalisée*. J'ai maintenant un bloc réutilisable que je peux insérer dans ce programme (ou dans n'importe quel autre futur programme) chaque fois que je souhaiterai faire avancer mon robot de 3 rotations du moteur, et le faire virer à droite de 90 degrés.

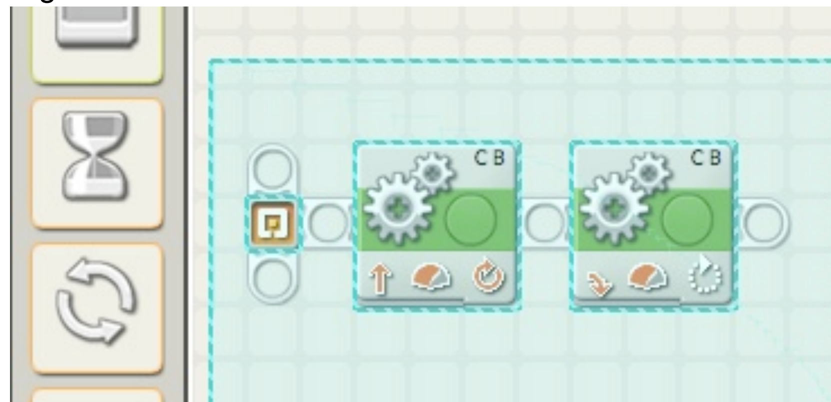
Les possibilités sont infinies. En utilisant ce concept, je peux créer de nombreuses collections de blocs NXT-G destinés à des actions spécifiques. Regroupés en paquets dans *Mes Blocs*, ils sont en permanence réutilisables.
Au bout d'un certain temps, votre collection s'enrichira, et de plus, vous économiserez du temps en évitant de réécrire certaines modules (comme faire un demi tour, avancer de 2 rotations, etc…).

Nous allons à présent aborder la création de ces blocs.

Création des Mon Bloc's:

La première étape consiste à écrire les actions *répétitives* qui seront regroupées dans un *Mon Bloc*.
1 - Créez un nouveau programme NXT-G. Dans l'espace de travail positionnez les blocs qui composeront une action. Dans notre cas, nous l'avons vu, 2 blocs DEPLACER (3 rotations + virage à droite de 90 degrés).
Cette action sera répétitive, donc réutilisable.
2 - Sélectionnez ces 2 blocs (plusieurs façons de procéder; la plus facile consiste à les inclure dans un rectangle de sélection).

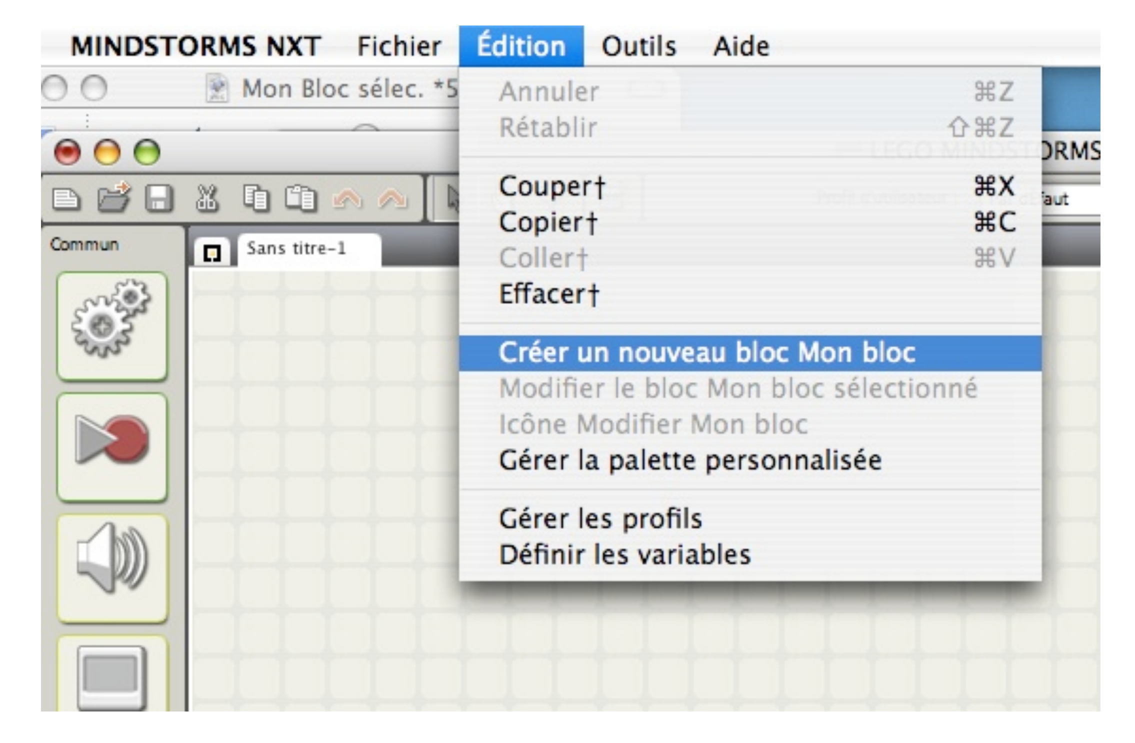
Figure 5



On peut aussi sélectionner chaque bloc en maintenant la touche MAJ appuyée.

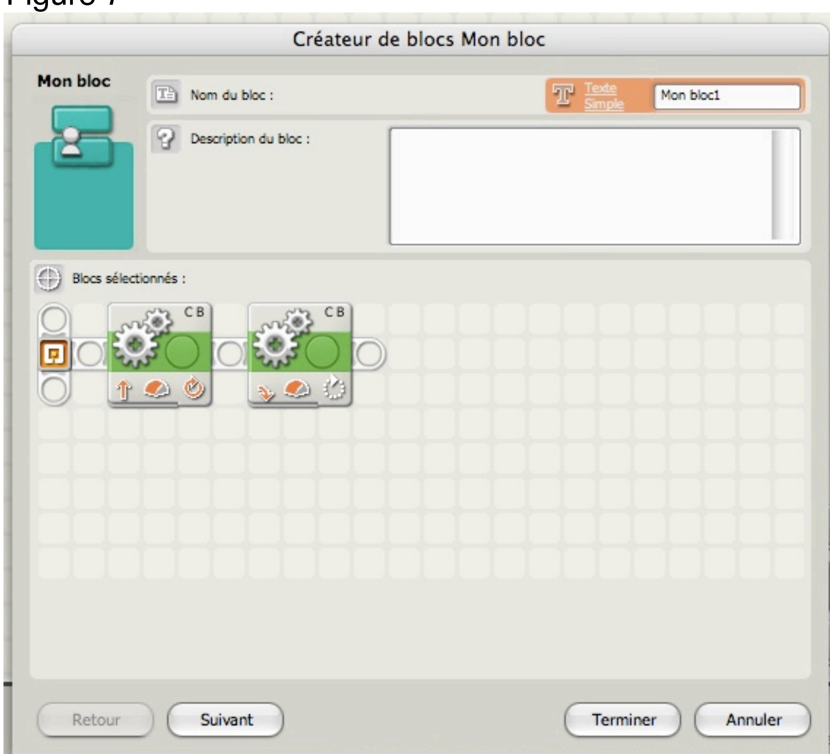
3 - A présent, dans la barre de menu, déployez "Edition" et choisissez "Créer un nouveau bloc Mon bloc".

Figure 6



Vous ouvrez une nouvelle fenêtre telle que ci-dessous:

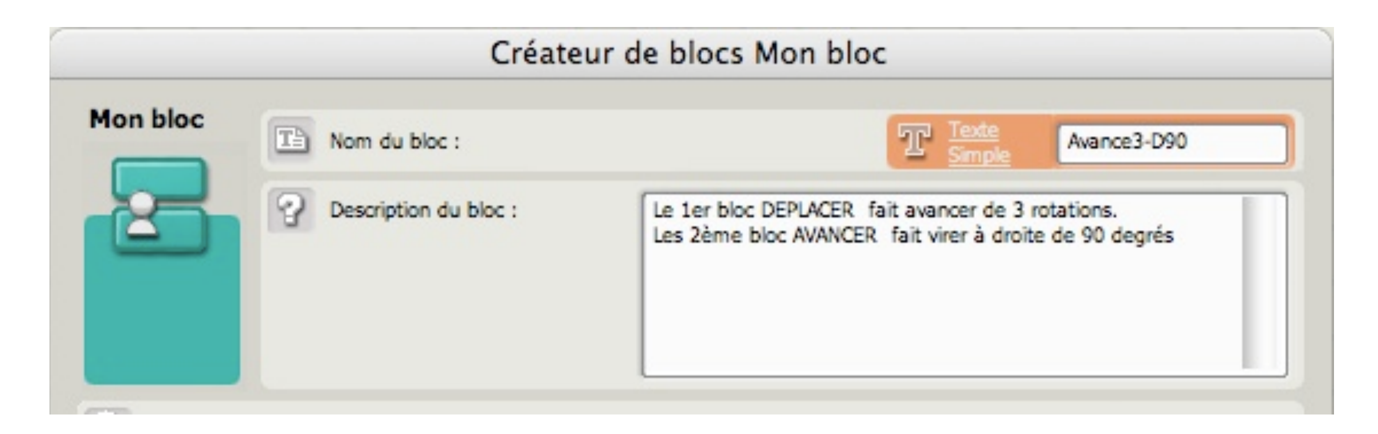
Figure 7



A noter dans la zone de travail la présence des 2 blocs sélectionnés.

Complétez les champs par des expressions facilement identifiables. Le nom du bloc est obligatoire; la description facultative.

Figure 8



Cliquez sur le bouton "Suivant" et choisissez une icône pour personnaliser votre nouveau Mon bloc, ou le bouton "Terminer" si vous ne voulez pas personnaliser l'icône.

Figure 9

Vous avez à votre disposition une collection d'icônes que vous pouvez dimensionner et associer à volonté. Il suffit de choisir l'un des motifs puis d'arranger selon vos goûts dans la zone d'édition, (carré blanc). Le résultat est visible en haut et à droite.
Quand vous jugez le travail terminé, appuyez sur le bouton "Terminer".

Votre nouveau *Mon Bloc* est maintenant accessible dans la "palette personnalisée" et utilisable comme n'importe quel autre bloc.

Figure 10



Ce nouveau bloc est venu s'ajouter à 2 anciens *Mes Blocs* déjà créés(dans le survol de la palette). Chaque fois que vous créerez un Mon bloc, il viendra s'ajouter à ceux existants. C'est ainsi que petit à petit la collection s'agrandit.

Remarque:

Tous ces blocs personnalisés sont stockés dans un dossier intitulé "Mes blocs" accessible à partir du menu "Edition" > "Gérer la palette personnalisée". On notera qu'ils sont eux-mêmes des petits programmes du type .rbt.

Fils et plots de données:

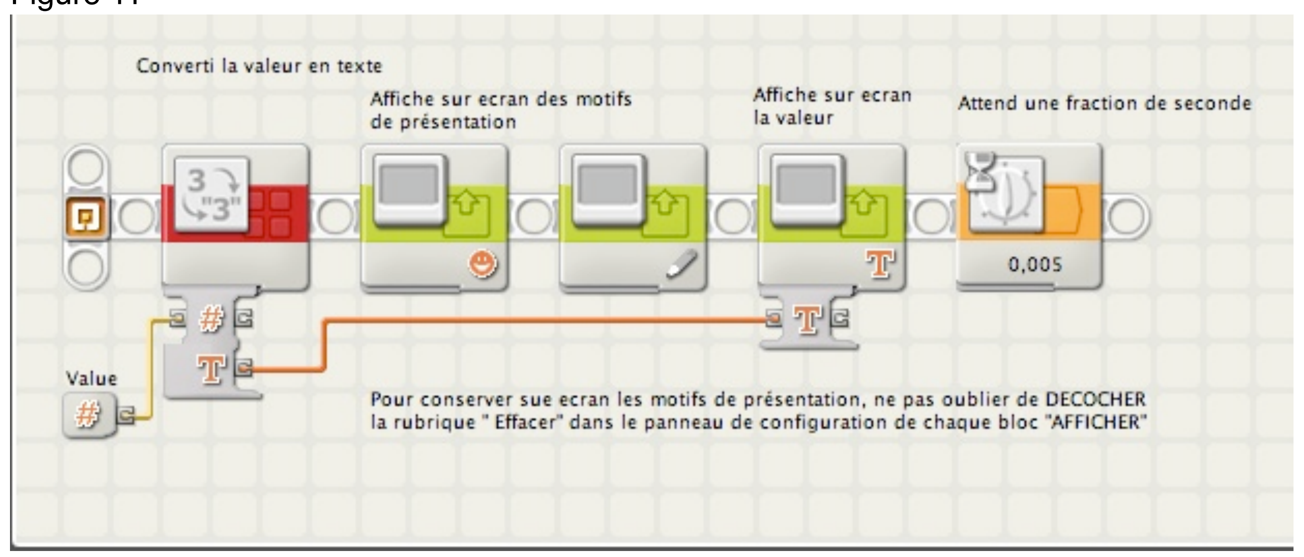
Au delà de ce qui vient d'être développé, il faut aussi noter que *Mes Blocs* peuvent recevoir et envoyer des données par le canal des fils de données.

Cela n'est possible que si les blocs sélectionnés pour constituer *Mes Blocs*, disposent de fils de données **préalablement** et **impérativement** configurés. Les ports d'entrée ou de sortie de ces blocs sélectionnés, laisseront apparaître des ports de données du *Mon Bloc* nouvellement créé. Cet aspect renforce leur puissance et offre une grande souplesse.

La manipulation des fils de données requiert une certaine habilité, et je vous conseille de vous exercer pour en déterminer leur opportunité.

Voici à titre d'exemple un *Mon Bloc* intitulé "Afficher Valeu"r:

Figure 11



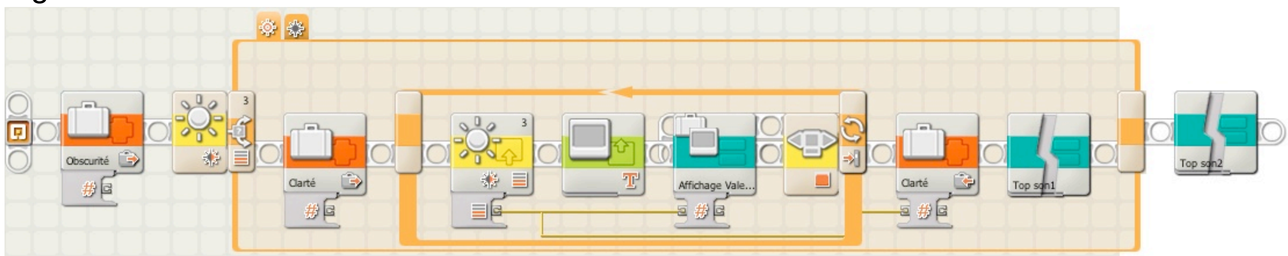
Ces blocs constituants un affichage (5 au total) ont été sélectionnés dans un programme plus complet qui répète à plusieurs reprises le même mode d'affichage.

Les plots et les fils de données ont été utilisés dans le corps du programme. En transformant cette suite en un *Mon Bloc* "Afficher valeur", le logiciel a ajouté en tête un plot indépendant "Value" bien visible dans cet exemple. La présence de ce plot "flottant"

signifie que ce *Mon Bloc* peut recevoir en entrée des données numériques. Tous autres types de données sont bien entendu acceptés (texte et logique).

Et voici une manière de l'utiliser dans un extrait de programme NXT-G plus général:

Figure 12



Remarque importante:
Vous noterez que le *Mon Bloc* intitulé "Top son1" est traversée par une bande zébrée, comme si l'icône était rompue. Cela signifie que le fichier de ce *Mon Bloc* ne figure pas dans la liste de *Mes Blocs* en cours d'utilisation.
Il est important de vous assurer que les *Mes Blocs* utilisés dans vos programmes, sont également stockés dans votre palette personnalisée. Si vous téléchargez ou copiez des programmes NXT-G, vous devez également télécharger ou copier les Mes Blocs qui les accompagnent (sauf s'ils figurent déjà dans la liste) et les installer avant le programme principal.

Quelques exemples utiles:

> Vous pouvez par exemple vous constituer une série de modules sonores composés à partir de blocs SON "Tonalité".

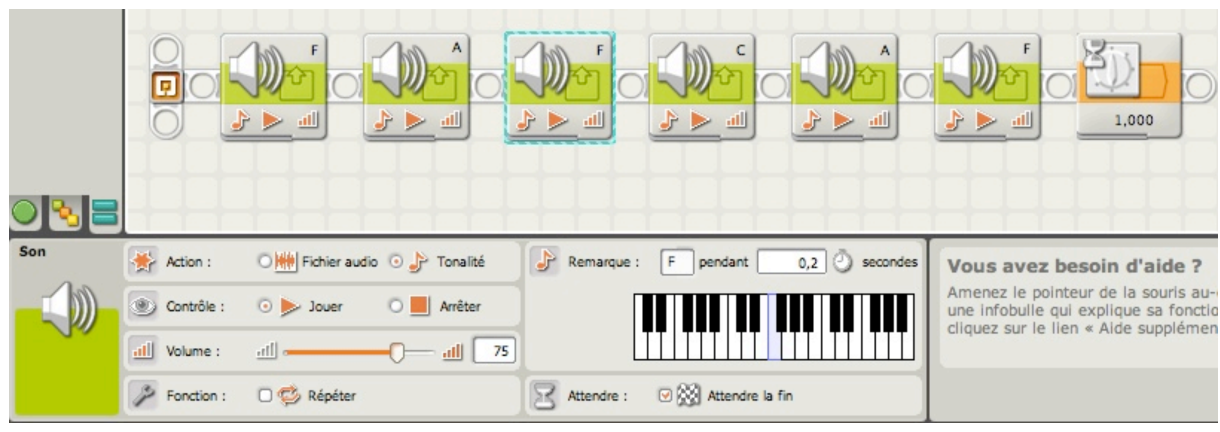


Figure 13



la séquence tonalité étant A, A, A et F

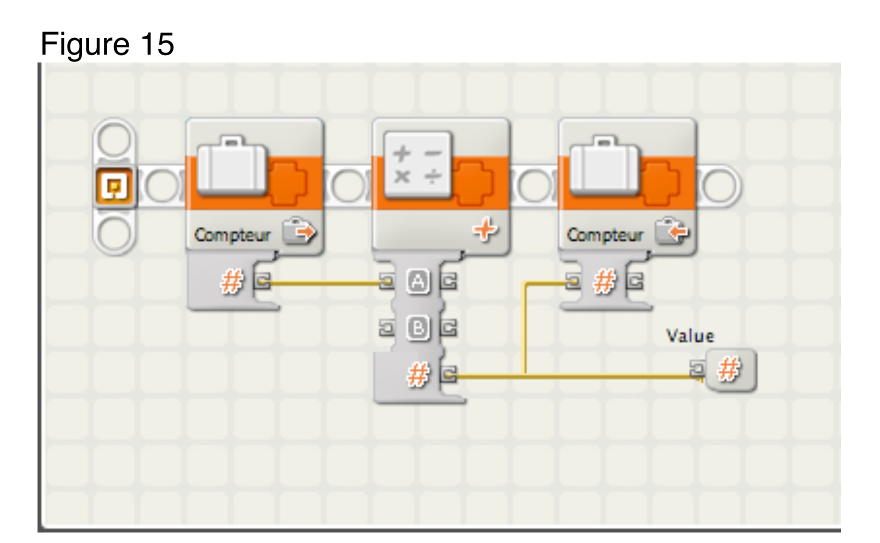
Figure 14



la séquence tonalité étant F, A, F, C,A et F
A vous maintenant de composer les airs musicaux de vos robots…

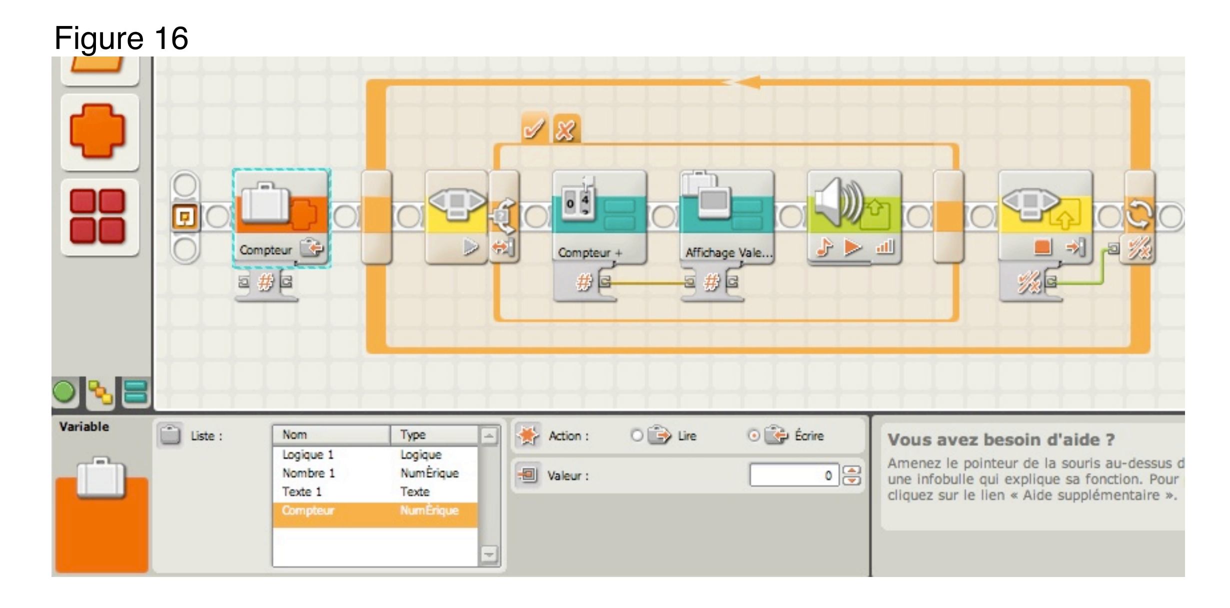
> Un compteur numérique d'incrémentation intitulé "Compteur +" combiné avec un bloc VARIABLE:

Figure 15



En association avec Mon bloc "Affichage valeur", permet une réduction notable de la longueur du programme:
Nécessite la création d'une variable "Compteur".

Figure 16



L'enrichissement des palettes est loin d'être terminé. Des développeurs et des constructeurs de capteurs NXT compatibles fournissent eux aussi des blocs spécifiques qu'il est aisé d'inclure dans le logiciel. Le NXT-G est loin d'avoir montré tout ce dont il est capable, et l'imagination des utilisateurs loin d'être tarie.
Exercez vous et n'hésitez pas d'expérimenter ces blocs; vous serez étonnés des résultats, aussi bien dans la diversité que dans la concision…



Introduction

Qu'est ce que Bluetooth ?

La traduction intégrale de cette expression signifie « dent bleue ». Quel rapport existe-t-il entre Bluetooth et la brique NXT ?

Pour répondre à cette question, commençons par un peu d'histoire.
Bluetooth est une spécification de l'industrie des télécommunications. Elle utilise une technologie sans fils, radio courte distance destinée à simplifier les connexions entre les appareils électroniques. Elle a été conçue dans le but de remplacer les câbles entre les ordinateurs et les imprimantes, les scanners, les claviers, les souris, les téléphones portables, les assistants personnels (PDA), les autoradios et les appareils photo numériques.
La technologie à été créée en 1994 par le fabricant suédois Ericsson, entreprise de télécommunications fondée en 1876. Cette marque est devenue (avec Sony Corporation) *Sony Ericsson*.

En 1998, Plusieurs grandes sociétés (Agere, IBM, Intel, Microsoft, Motorola, Nokia et Toshiba) s'associent pour former le Bluetooth Special Interest Group (SIG).
Ce nom est directement inspiré du roi danois Harald Ier (an 940) surnommé Harald Blåtand (« à la dent bleue »),

connu pour avoir réussi à unifier les États du Danemark, de Norvège et de Suède. Le logo de Bluetooth, est d'ailleurs inspiré des initiales en alphabet runique de *Harald Blåtand*.
En 2008 Le SIG, pour fêter les 10 ans de Bluetooth, annonce la prochaine génération de cette technologie sans fil, qui sera capable d'assurer des débits cent fois supérieurs à la version actuelle, passant donc de 1 Mb/s à 100 Mb/s (soit 12,5 Mo/s). Cette technologie - utilisée dans les téléphones mobiles, périphériques informatiques et autres appareils portables comme les assistants personnels (PDA) - va voir sa vitesse de transmission augmenter dans les années à venir, lui permettant alors d'être utilisée pour les vidéos haute définition et l'échange de fichiers avec un baladeur MP3 par exemple. La nouvelle norme incorporera une technologie radio, connue comme l'ultra wideband ou UWB.

Quelques explications

Revenons en arrière de plusieurs années, au port série ; notre bon vieux port série.
Il a été inventé dans les années 60 pour permettre de relier des périphériques, claviers, terminaux, matériels de mesures à des ordinateurs: le RS232.
Depuis, ce standard de l'EIA (Electronic Industries Association) a été rebaptisé en 1991 EIA232 et quelques modifications y ont été apportées. Les ports PS2 et claviers sont tous deux des ports série simplifiés. Le RS232 a été utilisé énormément pendant de nombreuses années et encore aujourd'hui, il est présent sur toutes les cartes mères.
Son évolution sans fil est l'IrDA, un protocole qui utilise les ondes lumineuses infrarouges pour transmettre des informations, la même technologie que dans nos télécommandes de télé. Ce protocole tout comme le RS232 ne permet que de connecter un seul périphérique à la fois. Un port série ne permet de ne relier qu'une souris ou qu'une imprimante (oui, il y a bien longtemps, les imprimantes se branchaient sur les ports séries) ou un autre périphérique. Moyennant bricolage matériel et logiciel, il est possible d'en relier plusieurs mais cela reste du bricolage.
C'est ainsi que l'USB (Universal Serial Bus) naquis. Pour permettre de brancher plusieurs périphériques en même temps sur le même port. Quand on regarde les spécifications de l'USB, on se rend d'ailleurs compte que ce n'est qu'un port série amélioré.
Restait donc à trouver l'équivalent de l'USB mais sans fil.
Bluetooth est l'évolution lointaine, sans fil du port série !
L'utilisation de périphériques Bluetooth est donc la même que celle des périphériques USB : relier un ordinateur à de petits périphériques personnels ou relier entre eux des périphériques personnels. Ce n'est donc pas une technologie permettant de monter des réseaux ou de relier entre eux de nombreux ordinateurs.
sur le site officiel Bluetooth http://www.bluetooth.com vous y trouverez beaucoup d'informations utiles.

Les fonctionnalités

Le NXT est donc équipé d'un dispositif Bluetooth, et pour *communiquer sans fil*, il a besoin d'un dispositif semblable installé chez ses correspondants.
Cela signifie entre autre, que l'ordinateur doit en être pourvu.
Dans le cas contraire, il est possible d'acquérir une clé spéciale qui se branche sur un port USB de l'ordinateur (acquisition sur le site de LEGO®).

En utilisant bluetooth vous pouvez échanger des programmes entre plusieurs NXT, vous pouvez établir des connections sans fils entre votre robot et votre ordinateur, tester les programmes immédiatement, même si votre robot se trouve à l'autre bout du local.
Si vous disposez d'un téléphone portable équipé d'un dispositif bluetooth, vous pouvez l'utiliser pour contrôler votre robot, et même l'employer comme un capteur perfectionné tel que capteur d'images.

Il existe diverses applications « mobiles » ; aussi je vous invite à consulter le site LEGO®, vous y trouverez des idées et des tutoriaux très utiles.
Pour résumer, les capacités du NXT se définissent ainsi :

♣ Peut-être connecté jusqu'à 3 appareils en même temps, mais ne communique qu'avec un seul appareil Bluetooth à la fois.
♣ Recherche et se raccorde aux autre appareils Bluetooth.
♣ Mémorise la connexion avec le dernier appareil pour un raccordement plus rapide.
♣ Rend le NXT « visible » ou « invisible » par d'autres appareils.
♣ La fonction Bluetooth peut-être désactivée pour économiser la batterie.

Compatibilité

Utilisateurs PC ou MAC:

Les informations de ce paragraphe proviennent du site LEGO® http://mindstorms.lego.com/Overview/Bluetooth.aspx.

LEGO MINDSTORMS NXT est compatible avec Bluetooth Software inclus dans Microsoft Windows XP Service Pack 2 et WIDCOMM Bluetooth Software pour Windows v 1.4.2.10 SPS ou plus récent.

MINDSTORMS NXT est compatible avec Bluetooth Software inclus dans Apple MacOS X 10.3.9, 10.4 et 10.5
Il est nécessaire de disposer de l'une de ces versions, pour établir une connexion entre votre ordinateur et la brique intelligente MINDSTORMS NXT.

Le tableau ci-dessous liste les appareils disposants de la technologie sans fil Bluetooth qui a été testé par LEGO® et ses partenaires.

Nota : ces appareils ont été testés seulement avec Widcomm® Bluetooth pour la version Windows plus récente que v. 1.4.2.10 SP5 et Bluetooth stacks inclus dans Microsoft Windows XP - Service Pack 2 / Apple MacOS X 10.3.9 et 10.4.

Tableau des compatibilités Bluetooth et MINDSTORMS NXT

Appareils Bluetooth	**Compatibilité**
Abe UB22S	***
Belkin F8T003 ver. 2 (short range)	***
BlueFRITZ! AVM BT adapter, BlueFRITZ! USB v2.0	***

Cables Unlimited USB-1520	***
Dell TrueMobile Bluetooth Module	*
Dell Wireless 350 Bluetooth Internal Card	***
Dlink DBT-120	***
MSI Btoes	***
MSI StartKey 3X-faster	***
TDK GoBlue	***
Qtrek, Bluetooth USB Adapter v2.0	***

Compatible / Bonne performance : ***
Non compatible / Maigre Performance : *

Mettre plusieurs NXT en communication

Vous avez peut-être envisagé de mettre votre robot NXT en communication ou en partage d'informations avec d'autres robots NXT. Ou alors, si vous possédez un deuxième kit NXT, vous envisagez d'associer une télécommande à votre robot, afin qu'il puisse commander cette seconde brique NXT.
Pour réaliser ce dispositif, vous devez utiliser un couple de blocs NXT-G que nous allons étudier dans cette leçon.
Certains trouvent ces blocs compliqués, mais en réalité ils sont simples, à condition de bien comprendre le concept de base de *l'expédition et de la réception des messages*.
Mais pour utiliser ce concept, il est nécessaire, d'abord, de mettre en communication via *Bluetooth* deux ou plusieurs briques NXT. C'est ce que je vais tenter de vous expliquer à présent.

1ère étape:
Vous allez d'abord travailler directement sur les menus internes du NXT et les boutons, en utilisant le petit écran.

a) Activation de bluetooth (si cette fonction n’est pas activée).

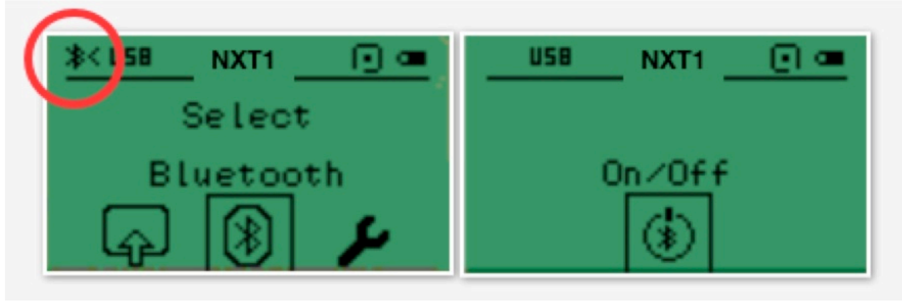
ce travail est à faire sur les deux NXT .
Allumez le premier NXT (le maître).



- à l'aide de la flèche droite gris clair, déplacez-vous dans les sous-menus jusqu'à BLUETOOTH . Sélectionnez BLUETOOTH en appuyant sur le bouton orange.
- Puis, sur ON/OFF en utilisant les flèche droite gris clair.



Pour activer Bluetooth, sélectionnez ON/OFF en appuyant sur le bouton orange. Quand vous sélectionnez ON, un petit symbole Bluetooth apparaît dans le coin haut à gauche de l'écran du NXT. (voir ci-dessous).



b) Renommez le NXT si nécessaire.

Pour identifier plus facilement vos NXT donnez leur des noms différents. A faire sur votre ordinateur en utilisant le logiciel Lego® Mindstorms® NXT-G.

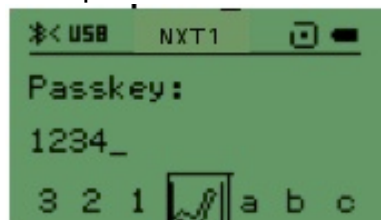
c) Mise en place d'une connexion Bluetooth NXT-à-NXT.

1. Assurez-vous que BLUETOOTH est activé sur les 2 NXT (voir ci-dessus)
2. Sur l'un des 2 NXT (le maître), déplacez-vous et sélectionnez le menu BLUETOOTH à l'aide des flèches gris clair.
3. Puis, choisir SEARCH. 

Activez SEARCH puis à l'aide des flèches gris clair choisir l'autre NXT (NXT 2). Cela demande à être fait par seulement l'un des deux NXT (le maître). 

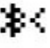
Acceptez NXT2. Vous serez alors invité à choisir un numéro de ligne de communication (le NXT peut être connecté à 3 appareils différents en même temps). Choisissez l'un des 3

numéros ; pour notre exemple n°2. 

Si cela est fait correctement, chaque NXT émettra un beep sonore et vous invitera à saisir un mot de passe. Ce mot est 1234 et doit être saisi automatiquement sur l’écran du NXT (à faire seulement pour la première fois). 

Sélectionnez la coche encadrée sur les deux NXT et cliquer sur le bouton orange pour établir la connexion. Cette Connexion peut être vérifiée en examinant l’angle haut à gauche de l’écran. Si le symbole Bluetooth n’est suivi que par le signe <, la communication n’est pas établie. Si par contre il est suivi par le signe







<>, alors la connexion est établie (voir cercle rouge)



Cette communication est "silencieuse", c'est à dire établie sans top sonore. Pour vous assurer qu'elle est bien active, il faut vérifier constamment les symboles entourés d'un cercle rouge. Après avoir établi la connexion, vous pourrez effectuer une vérification en choisissant CONNECTIONS du sous-menu BLUETOOTH ; les 2 NXT devraient être associés.

2ème étape:

Assurez-vous que les 2 robots NXT sont câblés correctement.

Assurez-vous que les programmes respectifs sont bien installés sur chaque NXT

Placez vos robots sur une surface d’évolution dégagée.

Lancez l’exécution des programmes correspondants sur chaque NXT, en commençant par le NXT esclave.

Remarque:

Pour de plus amples explications, je vous invite à consulter le menu "Aide et support" du logiciel NXT-G dans les rubriques *"Bloc Envoyer message"* et "*Bloc Recevoir message"*.

Les blocs "Message"

Pour communiquer entre eux, 2 NXT s'envoient et reçoivent des messages. Celui qui envoie est considéré comme "maître" et celui qui reçoit comme "esclave".
Le NXT (maître) qui communique avec d'autres NXT, par l'intermédiaire de canaux de communications, choisit le numéro de connexion (1, 2 ou 3) sous lequel le second NXT (esclave) est identifié. Sur le NXT (esclave), le NXT (maître) est identifié sous le numéro 0.
Numéro de connexion
Le numéro de connexion de chaque NXT peut être considéré comme son adresse dans l'univers sans fil. On peut donc envoyer un message à un NXT précis en indiquant son numéro de connexion.
Remarque:
Il n'y a pas de différence fondamentale entre un maître et un esclave. Tous deux disposent de bluetooth pour communiquer, mais il est judicieux de confier le rôle de maître à celui qui a le plus de responsabilités. Un exemple est celui d'un robot qui envoie un signal à un autre robot pour le déplacer, ou pour qu'il émette un son ou encore pour relever une valeur d'un capteur.
Parfois plusieurs robots peuvent évoluer à l'aide d'un même programme et s'envoient des signaux dans les deux sens; dans ce cas, choisissez l'un d'entre eux comme maître, et les autres (à concurrence de 3) seront configurés comme esclaves. Comme vous disposez de 4 valeurs possibles, les 3 briques NXT seront en communication avec une brique NXT maîtresse. Ainsi, un robot NXT pourra en contrôler jusqu'à 3.

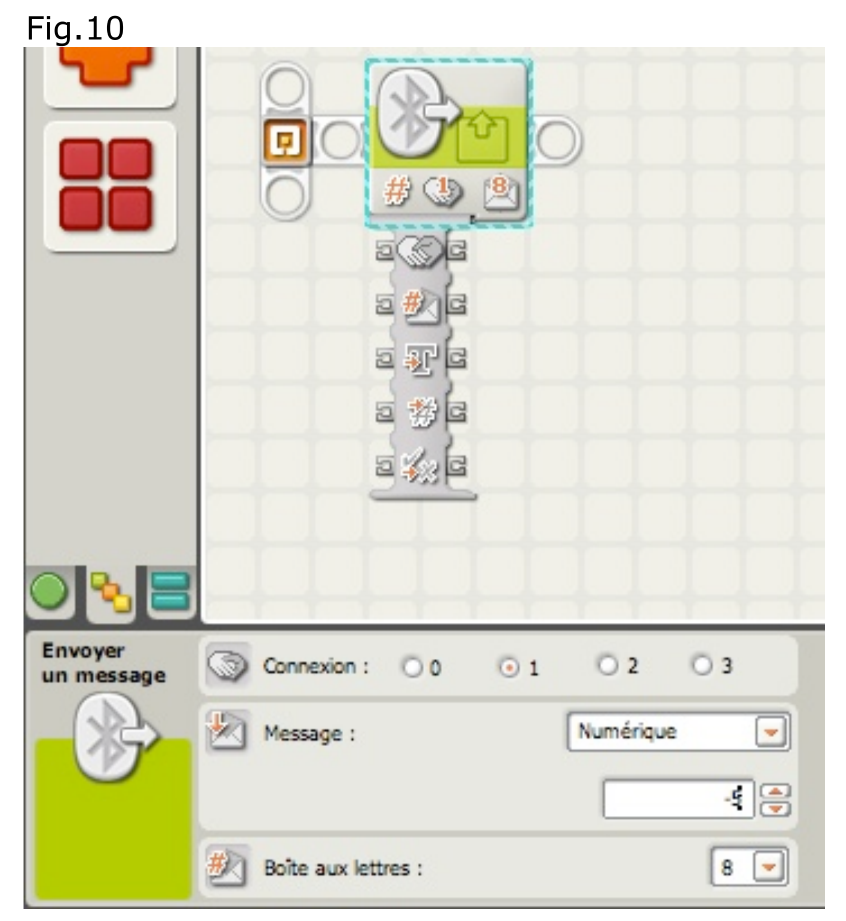
Un NXT dispose donc d'un moyen très intéressant pour adresser une simple information à un autre NXT. Par exemple relever le nombre d'activations du capteur tactile, et l'envoyer à un autre NXT. Et s'il était nécessaire d'envoyer plus d'informations? Par exemple la valeur du niveau de luminosité du capteur photosensible ainsi que que le nombre de rotations du moteur B à un autre robot. Cela est-il possible?
Si on admet que chaque message potentiel se comporte comme une *variable*, alors la réponse est oui.
Les blocs "Envoyer un message" et "recevoir un message" peuvent échanger entre eux jusqu'à 10 messages à l'aide de *boîtes aux lettres*.

Numéro de boîte aux lettres
Chaque NXT possède 10 numéros de boîtes aux lettres où des messages sans fil peuvent être déposés. Chaque numéro de boîte aux lettres peut contenir 5 messages. Si un numéro de boîte aux lettres contient déjà 5 messages, l'ajout d'un message supplémentaire forcera le NXT à effacer le plus ancien.

Bloc "Envoyer un message"

Ce bloc est disponible dans la "palette entière" série "action". C'est un bloc actif, émetteur comme le bloc son ou moteur.

Fig.10



L'icône présente une flèche sortante précisant qu'il s'agit d'un départ.

Pour envoyer un message sans fil (lorsque tous les NXT ont reçu un numéro de connexion), sélectionner tout d'abord le numéro de connexion du NXT de destination. Choisir ensuite le format du message (texte, numérique ou logique). Entrer le message en tapant du texte ou un nombre, ou en choisissant une condition logique (vrai ou faux). Enfin, choisir le numéro de boîte aux lettres (où le message sera stocké) sur le NXT de destination. Le format du message et le message proprement dit peuvent également être définis à l'aide de fils de données en entrée.

1. Utiliser les cases d'option pour choisir le numéro de connexion du NXT de destination.
2. Le menu déroulant permet de sélectionner le type (Texte, Numérique ou Logique).
3. Entrer le message en tapant du texte ou un nombre (selon le format choisi Texte ou Numérique), ou utiliser les cases d'option pour choisir une valeur logique (vrai ou faux).
4. Choisir le numéro de boîte aux lettres sur le NXT de destination.

Le bloc *Envoyer message* peut être contrôlé de manière dynamique en connectant à son plot de données des fils de données (provenant des plots de données d'autres blocs).

Fig.11

Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Envoyer message :

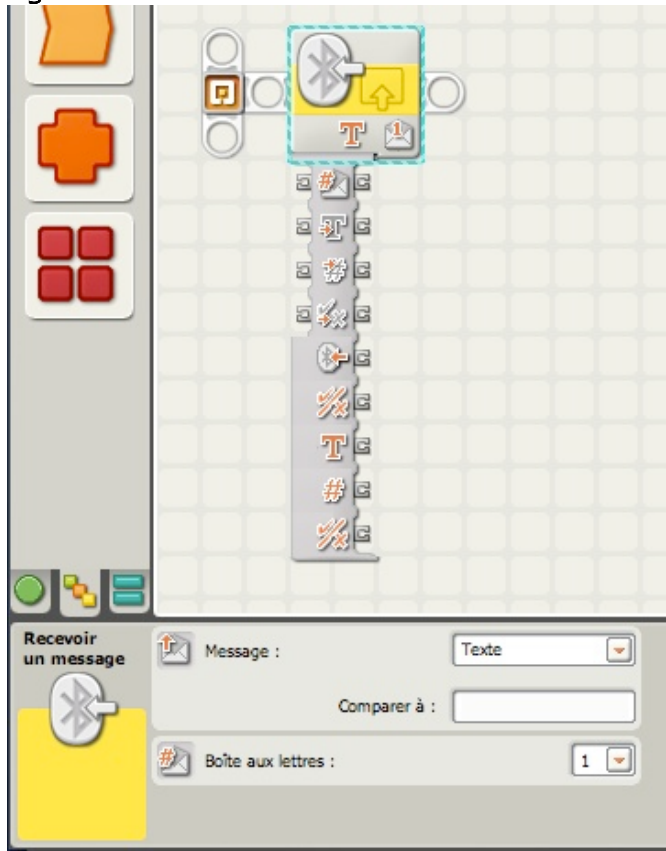
	Prise	**Type de données**	**Plage admise**	**Signification des valeurs**	**Prise ignorée quand...**
	Connexion	Numérique	0 - 3	Numéro de connexion de destination du message. Si vous choisissez « 0 », les messages seront envoyés du NXT esclave au NXT maître.	
	Boîte aux lettres	Numérique	1 - 10	Boîte aux lettres du destinataire où le message sera placé	
	Texte	Texte	58 caractères au maximum	Message à envoyer	Le type de message n'est pas Texte
	Numérique	Numérique	-2147483648 - 2147483647	Message à envoyer	Le type de message n'est pas Numérique
	Logique	Logique	Vrai/Faux	Message à envoyer	Le type de message n'est pas Logique

Bloc "Recevoir un message"

Ce bloc est disponible dans la "palette entière", série "capteur". Il fait donc partie de cette famille et se comporte comme un conteneur de messages, un peu comme un bloc *Variable*.

Fig.12





L'icône présente une flèche entrante précisant qu'il s'agit d'une arrivée.

Pour recevoir un message sans fil, fixer le type et le numéro de boîte aux lettres afin qu'ils correspondent à ceux du NXT source. La sortie de ce bloc peut être le message entrant ou une valeur logique « vrai/faux » (si on compare le message entrant à un message de test).

Réception d'un message

Pour recevoir un message sans fil (lorsque tous les NXT ont reçu un numéro de connexion), indiquer tout d'abord le format du message entrant (texte, numérique ou logique).
Si vous voulez définir un test qui vérifie si un message précis a été reçu (ce qui amène le bloc Recevoir message à envoyer un signal « vrai »), entrez le texte ou le nombre de test dans la zone de saisie, ou sélectionnez l'option Vrai ou Faux. Si le message entrant correspond au message de test, le bloc envoie un signal « vrai » via sa prise « √/x ».

1. Le menu déroulant permet de sélectionner le type (Texte, Numérique ou Logique) du message reçu.
2. Si on souhaite comparer le message entrant à un message de test, entrer le texte ou le nombre de test (si vous avez respectivement choisi le format Texte ou Numérique), ou utilisez les cases d'option pour choisir la valeur logique (vrai ou faux) du test.
3. Choisissez le numéro de boîte aux lettres où le message entrant sera stocké.

Le bloc *Recevoir message* peut être contrôlé de manière dynamique en connectant à son plot de données des fils de données (provenant des plots de données d'autres blocs).

Fig.13

Le tableau ci-dessous présente les différentes caractéristiques des prises du plot de données du bloc Recevoir message :

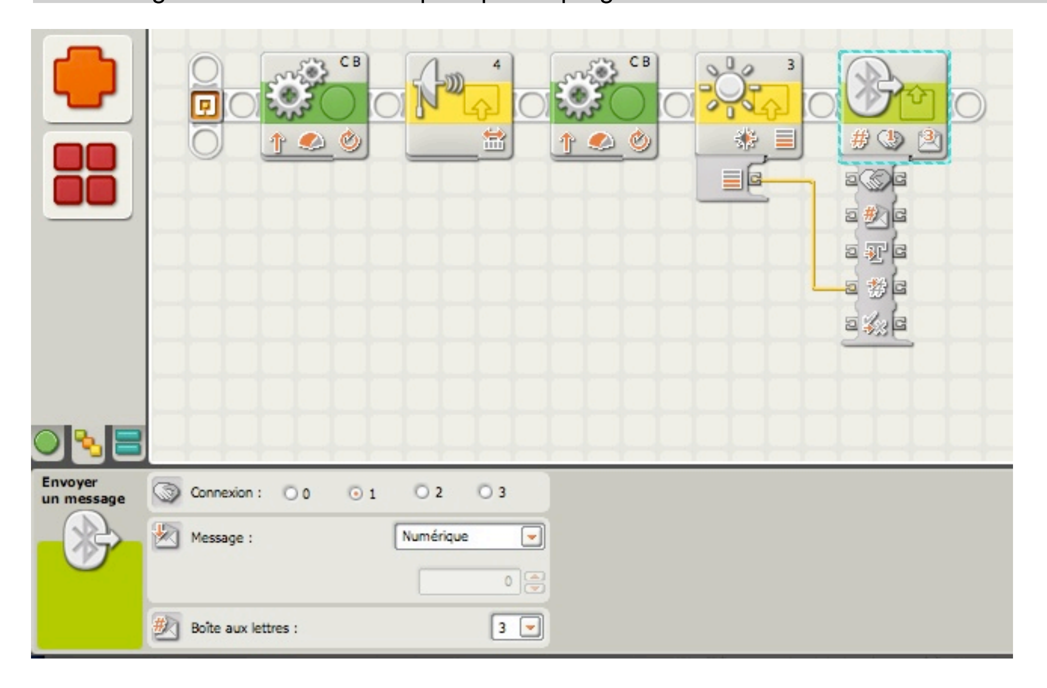
	Prise	Type de données	Plage admise	Signification des valeurs	Prise ignorée quand...
	Boîte aux lettres	Numérique	1 - 10	Boîte aux lettres à consulter	
	Texte en entrée	Texte	58 caractères au maximum	Valeur de référence de la comparaison	Le type de message n'est pas Texte
	Nombre en entrée	Numérique	-2147483648 - 2147483647	Valeur de référence de la comparaison	Le type de message n'est pas Numérique
	Logique en entrée	Logique	Vrai/Faux	Valeur de référence de la comparaison	Le type de message n'est pas Logique
	Message reçu	Logique	Vrai/Faux	Vrai si un message est reçu (c'est-à-dire que la boîte aux lettres n'est pas vide)	
	Oui / Non	Logique	Vrai/Faux	Résultat de la comparaison	
	Texte en sortie	Texte	58 caractères au maximum	Données du message	Le type de message n'est pas Texte
	Nombre en sortie	Numérique	-2147483648 - 2147483647	Données du message	Le type de message n'est pas Numérique
	Logique en sortie	Logique	Vrai/Faux	Données du message	Le type de message n'est pas Logique

Voici à présent un exemple, pour illustrer ce dispositif d'envoi et de réception de messages.
Commençons par le pseudo code:

LEO > BONG, avance jusqu'à ce que le capteur US détecte un objet à 25 cm, puis arrête-toi.
LEO > BONG, lit la valeur de ton capteur photosensible.
LEO > BONG, envoie la valeur relevée de ce capteur à BONGbis, en utilisant la boîte aux lettres 3.

Voici à quoi ressemble le programme écrit en NXT-G:

Fig. 14



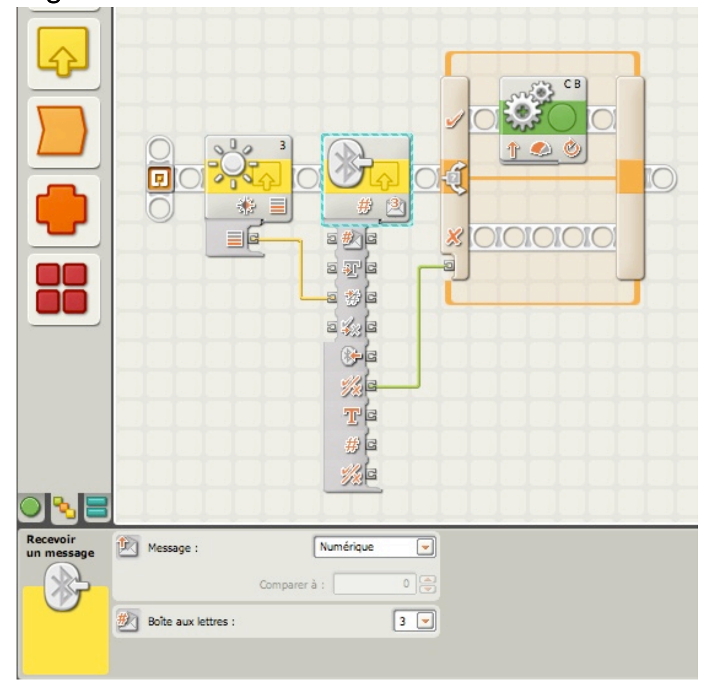
La valeur du capteur photosensible est adressé au bloc *Envoyer un message* dans la *boîte aux lettres* 3, à l'aide d'un fil de données reliant le plot *Intensité* du capteur photosensible au plot *numérique* du bloc *Envoyer un message*.

Maintenant, si je dispose d'un second robot appelé BONGbis, je peux imaginer un autre pseudo code:

LEO > BONGbis, vérifie la valeur numérique déposée dans ta boîte aux lettres 3.
LEO > BONGbis, compare cette valeur à celle lue par ton capteur photosensible
LEO > BONGbis, si ces deux valeurs sont égales, avance de 8 rotations.

Et la traduction en programme NXT-G est la suivante:

Fig.15



On observe que BONGbis relève la valeur de son capteur *photosensible* et cette valeur est transmise au bloc *Recevoir un message*. Puis, BONGbis vérifie et lit le contenu de sa boîte aux lettres 3. Une comparaison est alors faite entre la valeur stockée et celle lue sur son capteur photosensible. Cette comparaison est faite automatiquement, parce qu'une valeur numérique a été fournie par le capteur au moyen d'un fil de données en *entrée*, relié au bloc *Recevoir un message*, et qu'un autre fil de données en *sortie logique* est relié au plot *logique* du bloc *Commutateur*. Cette disposition force le bloc *Recevoir un message* à comparer ces 2 valeurs. Si ces 2 valeurs sont égales, un valeur logique VRAI est alors transmise au bloc Commutateur par le plot OUI/NON (à ne pas confondre avec le *plot logique en sortie*), qui sélectionne le chemin correspondant; c'est à dire le déplacement de BONGbis (8 rotations).

Les blocs Envoyer un message et Recevoir un message sont absolument nécessaires pour que 2 robots NXT ou plus, puissent communiquer et échanger des informations.
Il est intéressant de signaler qu'avec un deuxième kit NXT, il est possible de réaliser une télécommande pour contrôler un autre robot. Pour cela vous utiliserez dans la télécommande, les blocs *Envoyer un message* qui adresseront des valeurs numériques au robot, comme le nombre de rotations ou de degrés aux moteurs B et C ou d'autres paramètres comme la vitesse ou la puissance.

NXT et le téléphone portable

Comment faire communiquer un téléphone portable avec un NXT?
Comment commander un NXT à partir d'un portable?
Nous voici devant quelques paris très intéressant qui offrent encore plus de possibilités à vos robots.

Contrôler le NXT de la paume de la main ! Télécommander un robot NXT comme un véritable racer LEGO®.
Contrôler quelques programmes personnalisés ou faire prendre des photos par un NXT en utilisant un téléphone portable.
Tout cela est possible à l'aide d'un NXT et d'un mobile. Et tout se passe comme si ce mobile se comportait comme un 2ème NXT (et il peut être maître ou esclave).

Télécharger le logiciel '*NXT Mobile Application*':

Pour qu'une liaison Bluetooth entre un portable et un NXT soit possible, il est nécessaire d'installer sur le portable un logiciel particulier conçu par LEGO®. Pour que cette installation soit possible, le portable doit être en mesure de supporter le téléchargement, c'est-à-dire être compatible.
Cette liste fournie par LEGO® contient les types d'appareils courants compatibles avec le logiciel *NXT Mobile Application*:
Nokia http://www.nokia.com/
6680
3230
Sony Ericsson http://www.sonyericsson.com
W800i
W550i
K610i
K800i

K750i
Z710i
Z550i
K510i
BenQ-Siemens http://communications.siemens.com/
CX75
S65

Ces appareils téléphoniques, ont été vérifiés par LEGO® et fonctionnent avec le logiciel *NXT MOBILE APPLICATION*.

D'autres appareils sont susceptibles de fonctionner, en particulier ceux qui disposent une zone réservée pour télécharger des applications et qui acceptent le langage Java.
Il faudra donc vérifier ce service en examinant le manuel de l'utilisateur du portable.

Le logiciel *NXT Mobile Application*

A télécharger à partir du site LEGO®
http://mindstorms.lego.com/eng/Overview/Mobile%20Application.aspx

Usage

Ce logiciel propose 2 manières pour contrôler le NXT: par télécommande ou par programmation.

En télécommande, il suffit de déplacer la molette d'orientation pour faire évoluer le robot.
En programmation, la molette d'orientation sert à sélectionner le bon programme, puis en appuyant sur un numéro, envoie le programme correspondant sur le NXT.

Procédure

1 - Vous devez pouvoir disposer de bluetooth sur votre ordinateur afin d'installer le logiciel *NXT Mobile Application* sur votre appareil téléphonique portable.
2 - Votre téléphone portable doit pouvoir fonctionner en bluetooth et interpréter le langage Java (JSR-82).
Assurez vous que votre opérateur vous alloue une zone réservée aux applications et que la configuration de votre appareil vous permet d'installer java. Vous devrez peut-être modifier la configuration pour utiliser les applications java; (consulter le manuel de l'appareil).
www.mindstorms.com/bluetooth/
Un tableau récapitulatif des appareils compatibles est inclus dans le paquet de téléchargement du logiciel.
3 - Le logiciel *NXT Mobile Application* est compatibles avec la brique NXT, versions FW 1.03, AVR 1.01, BC4 1.01 et suivantes.

Téléchargement et installation

1 - Assurez-vous que votre téléphone portable est en mesure de fonctionner avec cette application.
2 - Téléchargez le logiciel *NXT Mobile Application* sur votre ordinateur.
3 - Assurez vous que la liaison Bluetooth est activée sur votre téléphone et le NXT
4 - Recherchez sur votre ordinateur le fichier *NXTmobile.jar* spécifique à votre appareil (ils sont rangés dans des dossiers par type et marque), puis installez le sur votre téléphone:
PC: rechercher l'application téléchargée puis clic droit sur le fichier. Choisir 'Send to' (envoyer à) dans le menu et choisir 'Bluetooth unit'. Trouver et sélectionnez votre appareil, puis suivez les instructions.

MAC: Click sur l'icône Bluetooth de la barre de menu, puis choisir 'Envoyer un fichier'. Sélectionnez le *NXTmobile.jar* de votre appareil, puis clic sur 'Envoyer'. Rechercher votre téléphone dans la liste des appareils et clic sur 'Envoyer'. Suivre ensuite les instructions.

Désinstallation du logiciel NXT Mobile Application
Se référer au manuel d'instructions de votre appareil téléphonique pour désinstaller le logiciel. En général, il suffit de supprimer le programme du mobile.

Se connecter

1 - lancer le logiciel *NXT Mobile Application* de votre portable en navigant vers le dossier le contenant (généralement 'games' ou 'Applications'). Dès le lancement, le programme cherchera automatiquement les appareils NXT. La première fois que vous connecterez un nouveau NXT, il sera nécessaire de 'jumeler' ce NXT et le logiciel *NXT Mobile Application:*
- Sur le NXT: accepter la connexion à l'aide de la coche (mot de passe)
- Sur le Mobile: Saisir le mot de passe (défaut du NXT étant 1,2,3,4)

La fois suivante, le logiciel trouvera automatiquement votre NXT; vous n'aurez alors, qu'à le sélectionner pour se connecter.
Astuce: personnalisez votre NXT et votre Mobile en leur donnant des noms distincts. Cela évitera toute confusion avec d'autres NXT ou Mobiles au moment d'une connexion Bluetooth.

Utilisation courante

Une fois le logiciel *NXT Mobile Application* lancé, vous disposez des options suivantes à partir du menu principal:
Info
Une brève description du logiciel *NXT Mobile Application*, où trouver de l'aide plus quelques informations - et la terminologie employée dans le logiciel.
Télécommande
Elle permet de contrôler 2 moteurs sur le NXT. Utiliser la molette d'orientation du portable pour avancer, reculer et arrêter. On peut également choisir de contrôler un moteur à la fois. Si votre robot est conçu comme un tribot, il pourra fonctionner comme un engin téléguidé.
Contrôle par programme
Ce mode vous permet de contrôler tous les programmes enregistrés sur le NXT. D'abord, choisir le programme à contrôler. Ensuite vous pouvez déclencher un ordre par un message envoyé à votre NXT, en appuyant sur une touche numérique du portable. Ce que fera le NXT après avoir appuyé sur une touche, dépend entièrement de votre programme.
Données enregistrées (Collected data)
Si le NXT est conçu pour que votre portable prenne des photos, c'est l'endroit où vous récupérerez les fichiers, aussi bien par exemple, que les relevés des capteurs.
Remarque: les données récupérées, ainsi que les photos sont supprimées de la caméra quand on quitte l'application.

Utilisation avancée et exemples de programmes

En complément des informations précédentes, la molette de navigation peut être complétée par les touches du portable:
[1] [4] [7] et [*] contrôle le moteur A
[2] [5] [8] et [0] contrôle le moteur B
[3] [6] [9] et [#] contrôle le moteur C

Ces touches permettent d'affiner les rotations des moteurs, bien mieux que la molette. Par exemple, faire suivre un trajet courbe par un véhicule, ou contrôler les 3 moteurs en même temps.

Exemple

Appuyer sur [3] active l'avancée du moteur C. Appuyer sur [6] fait démarrer le moteur C et en répétant le geste sur ce même bouton accroît la vitesse. Appuyer sur [9] fera décroître la vitesse. Pour arrêter le moteur, appuyer sur [3].

Pour activer la marche arrière du moteur C, appuyer sur [#]. Appuyer à nouveau sur   [6] augmentera la vitesse et [9] diminuera cette vitesse en marche arrière.

Appuyer sur [3] stoppe le moteur.

Astuce: pour arrêter et réinitialiser les moteurs, appuyer sur OK de la molette de navigation.

Vous pouvez également changer les paramètres de défaut en choisissant "Options" dans le mode "télécommande (remote control)". La valeur de défaut est moteurs B +C, mais si votre robot est branché différemment, le changement est réalisé en utilisant ce mode.

Contrôle par programmation

Lorsque le logiciel *NXT Mobile Application* démarre, il recherche tous les programmes de votre NXT de manière à les rendre accessibles. Vous pouvez alors activer l'un d'entre eux, en sélectionnant dans le menu "options" du portable, ou utiliser ces raccourcis:

Sur la molette de navigation: Up, Down, Left, Right

Sur le clavier: [*] [+] [#]

Pour personnaliser les raccourcis:

1 - choisir "Options"

2 - sélectionner le programme concerné et appuyer sur "More" puis sur "Move Prg"

3 - Sélectionner la touche à associer au programme et choisir "More" puis sur "Place Prg"

Quand un programme est activé, des messages peuvent être adressés en appuyant sur les touches numériques de [0] à [9].

Exemple:

A télécharger à partir du site LEGO®

(NXT Mobile Application Documentation.zip (zip 1MB))

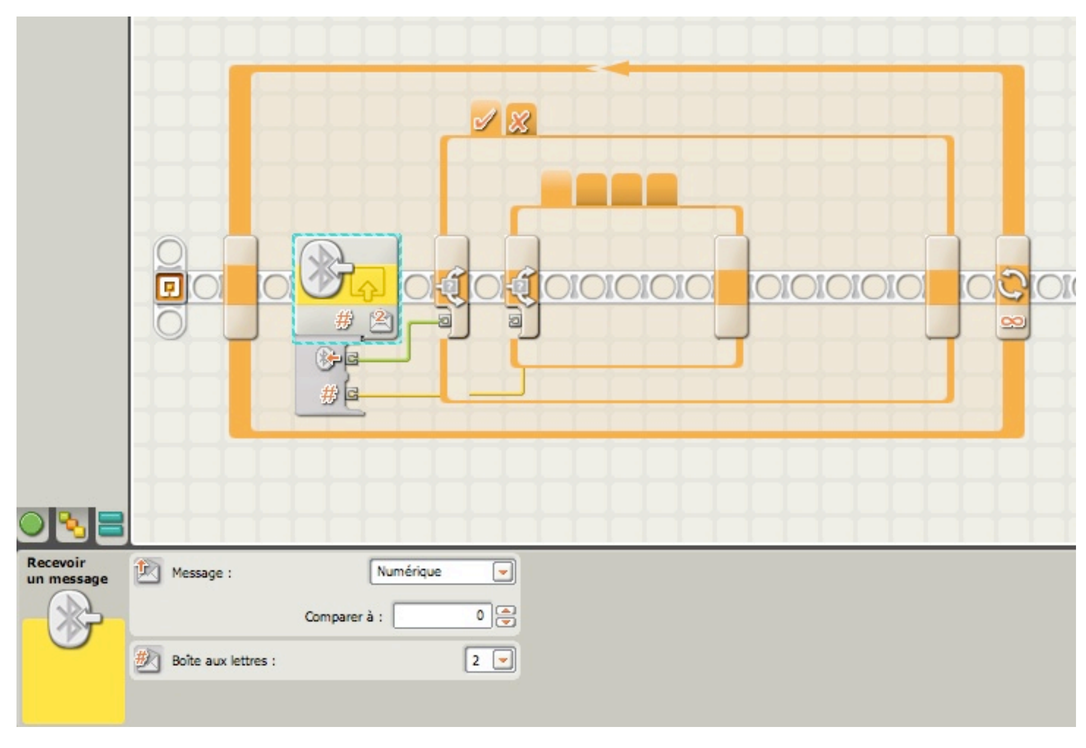
Ouvrir à partir de votre ordinateur, les 2 programmes tests en utilisant le logiciel NXT-G, puis installez les sur votre NXT (vous savez comment faire, n'est-ce pas?).

SendMsg.rbt

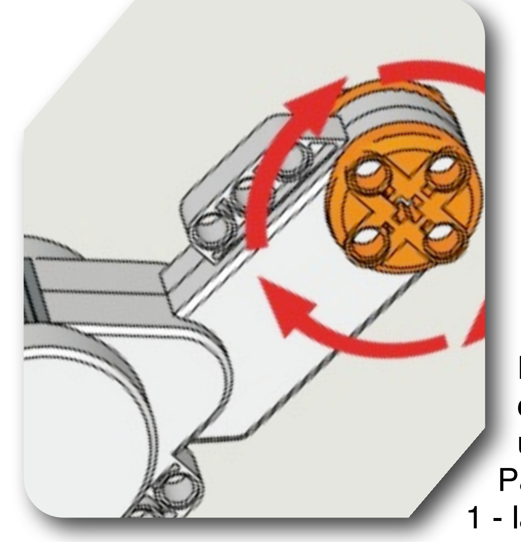


Raccorder le capteur tactile au port 1? Quand le capteur est appuyé, le NXT adresse un message à votre portable pour prendre une photo (si votre potable dispose de cette fonction).
Raccorder ce même capteur au port 2 et appuyer. Cela déclenche une sonnerie sur votre portable.
Raccordé au port 3, appuyer sur le capteur déclenche l'envoi d'un message texte vers le portable: "hello".
Remarque: vous pouvez retrouver les données reçues par le portable dans "Collected data".

RecieveMsg.rbt



En appuyant sur les touches [1] [2] ou [3] du portable, fait dire par le NXT le numéro activé.
Remarque: essayez de personnaliser ces 2 programmes tests pour en savoir plus sur la manière de rendre les programmes compatibles avec le logiciel NXT Mobile Application.





Leçon n°13: TRUCS et ASTUCES... PROGRAMMES UTILES

Pour programmer votre robot, vous utilisez parfois certaines relations mathématiques qui conditionnent une parfaite exécution du déplacement.
Parmi ces relations, les deux les plus employées sont:
1 - la conversion d'un angle (exprimé en degrés) en rotations, c"est à dire en nombre de tours du moteur,
2 - le calcul d'une distance à parcourir à partir d'un angle exprimé en degrés ou en nombre de rotations.
Les utiliser évite des manipulations et corrections successives, ou du moins les limite quand on recherche une certaine précision. C'est ce que nous allons examiner.

Par ailleurs, nous savons que le petit écran du NXT permet l'affichage de textes, de formes élémentaires et de symboles graphiques. Ce que nous savons moins bien, c'est que cet écran permet une certaine animation, grâce à l'utilisation des coordonnées cartésiennes en x et y. C'est ainsi, par exemple, que l'on peut associer une variable à une barre de progression mobile en fonction des différentes valeurs de cette variable. Mais pour réaliser ce travail, il nous faut connaître les caractéristiques et les conventions adoptées pour cet écran. C'est ce que nous verrons par la suite.
Enfin, au fur et à mesure que votre bibliothèque de programmes s'agrandit, vous constaterez que certains modules reviennent souvent et sont réutilisables pour des projets différents. D'où, l'idée de les écrire une fois pour toute pour les réutiliser à l'occasion.

Déplacements du robot

Retenez d'abord ces quelques formules, qui expriment les relations entre degrés et rotations.
Quand il s'agit de choisir les paramètres de durée dans le panneau de configuration du bloc MOTEUR ou DEPLACER, les 2 éléments les plus souvent utilisés sont l'angle (en degrés) ou le nombre de rotations. La durée en secondes est plus rarement choisie, parce que les distances parcourues en 5 secondes par exemple, ne seront jamais identiques du fait de la charge de la batterie qui diminue selon l'usage.

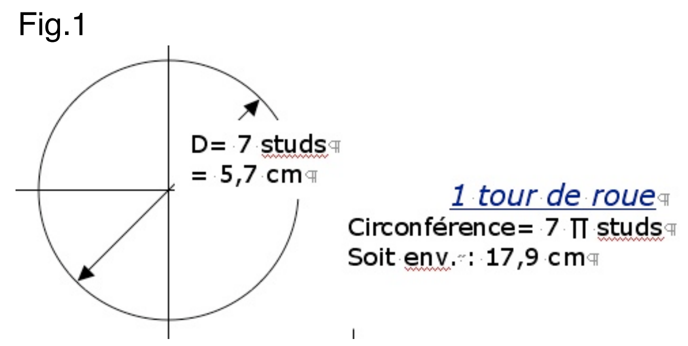
Les formules sont simples et faciles à retenir:

DEGRES = (NOMBRE DE ROTATIONS) x 360
ROTATIONS = (NOMBRE DE DEGRES) / 360
Attention: il s'agit du nombre de degrés ou de rotations du MOTEUR et non des roues du robot.
Ces formules de conversion permettent de modifier un programme selon les circonstances et les préférences de chacun.

Comment à présent convertir les degrés ou rotations en DISTANCE?

La première chose à faire avant de déterminer la distance à parcourir, c'est de convertir la *durée* en nombre de rotations (voir la formule ci dessus). Si la durée de votre bloc DEPLACER est exprimée en degrés, il suffit de choisir le paramètre *rotation* dans le panneau de configuration; le calcul est fait automatiquement.
Exemple: supposons que la durée soit de 900 degrés; le nombre de rotations correspondant est de 900/360 = 3,5. Le résultat est exprimé en valeur décimale.
Ensuite, nous avons besoin d'une deuxième valeur: celle du diamètre de la roue.
Il s'agit en l'occurence du diamètre extérieur de la roue, y compris son pneu.

Fig.1

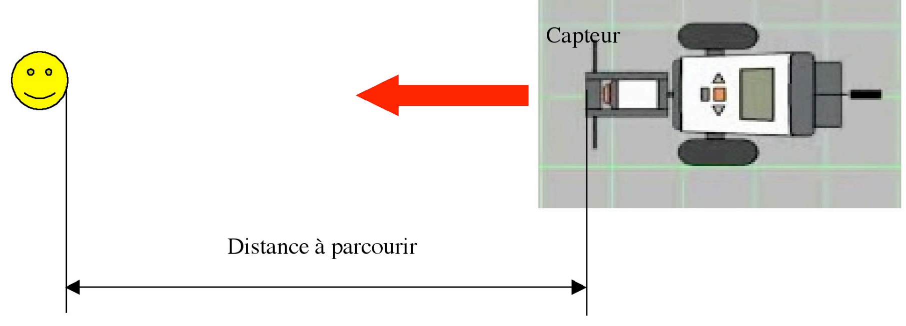


La circonférence est égale à:
D (diamètre) x 3,14 (pi).
Si vous prenez l'une des roues du NXT, vous constaterez que le diamètre, pneu compris, est approximativement de 5,7 cm.
La valeur de pi a été volontairement limitée à 2 chiffres après la virgule, aussi la circonférence (égale à un tour de roue) s"établit à 5,7 x 3,14 = approximativement 17,9 cm.
En reprenant notre exemple de 900 degrés, la distance parcourue est:
17,9 cm x 3,5 = 62,65 cm approximativement.

Remarque: La distance parcourue est donc fonction du diamètre de la roue. Plus la roue est grande et plus la distance parcourue l'est également.

Comment déterminer avec suffisamment de précision une distance à parcourir?

Il existe une méthode qui évite tous ces calculs manuels. N'oubliez pas que le NXT est aussi un calculateur et que chaque moteur est également un capteur grâce à son tachymètre interne. Nous allons utiliser ces particularités en les associant aux blocs NXT-G.
Fig.2



<u>Première méthode:</u>

1. Ouvrir un fichier en cours d'élaboration
2. Allumer le robot

3. Brancher le robot au PC (câble USB ou bluetooth)
4. Positionner le robot à son point de départ
5. Sélectionner un bloc MOTEUR dans le programme
6. Dans le panneau de configuration sous l'icône MOTEUR cliquer sur le bouton « Réinitialiser ». La valeur zéro (0) apparaît dans le champ situé au dessus

Fig.3



7. faire avancer à la main le robot jusqu'au point de contact désiré
8. Noter la valeur qui s'affiche dans le champ: ici 1085.

Fig.4



9. Noter la couleur des caractères (noir, sens positif ; rouge, sens négatif)
10. Reporter cette valeur dans la zone « Durée » du panneau de configuration (en degrés).

Remarque : cette méthode dispense de calculer la distance en fonction du diamètre des roues.

Deuxième méthode:

Procéder de la même manière si vous ne souhaitez pas travailler sur un programme existant.

1. Créer un nouveau programme
2. Allumer le robot
3. Brancher le robot au PC (câble USB ou bluetooth)
4. Positionner le robot à son point de départ
5. Placer un bloc DEPLACER au début du programme
6. Dans le panneau de configuration sous l'icône DEPLACER cliquer sur le bouton "R" (Réinitialiser). La valeur zéro (0) apparaît dans les champs B et C des moteurs synchro situés au dessous

Fig.5



7. faire avancer à la main le robot jusqu'au point de contact désiré
8. Noter la valeur qui s'affiche dans les champs B et C: ici 784 et 890.
   Faire la demi somme soit (784+890)x0,5= 837

Fig.6

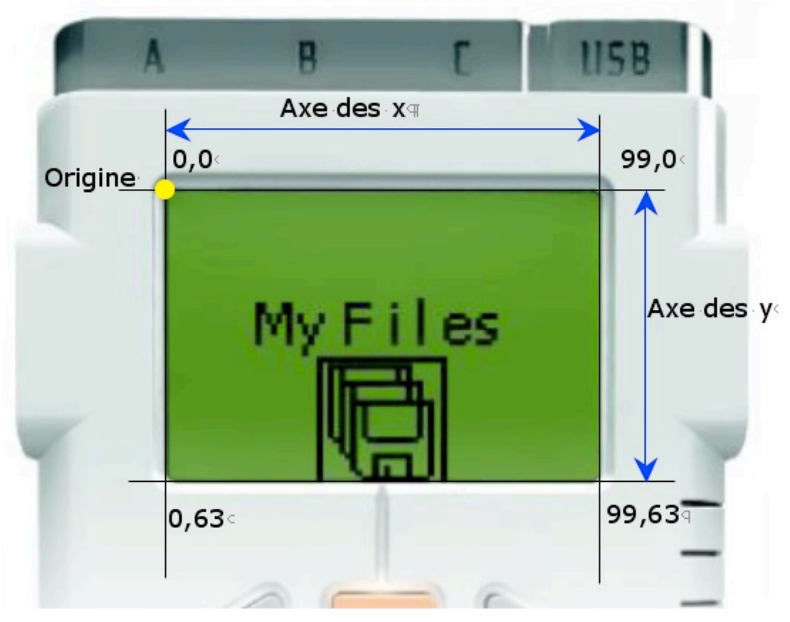


9. Noter la couleur des caractères (noir, sens positif ; rouge, sens négatif)
10. Reporter la valeur 837 dans la zone « Durée » du panneau de configuration (en degrés).

Utilisation de l'écran NXT

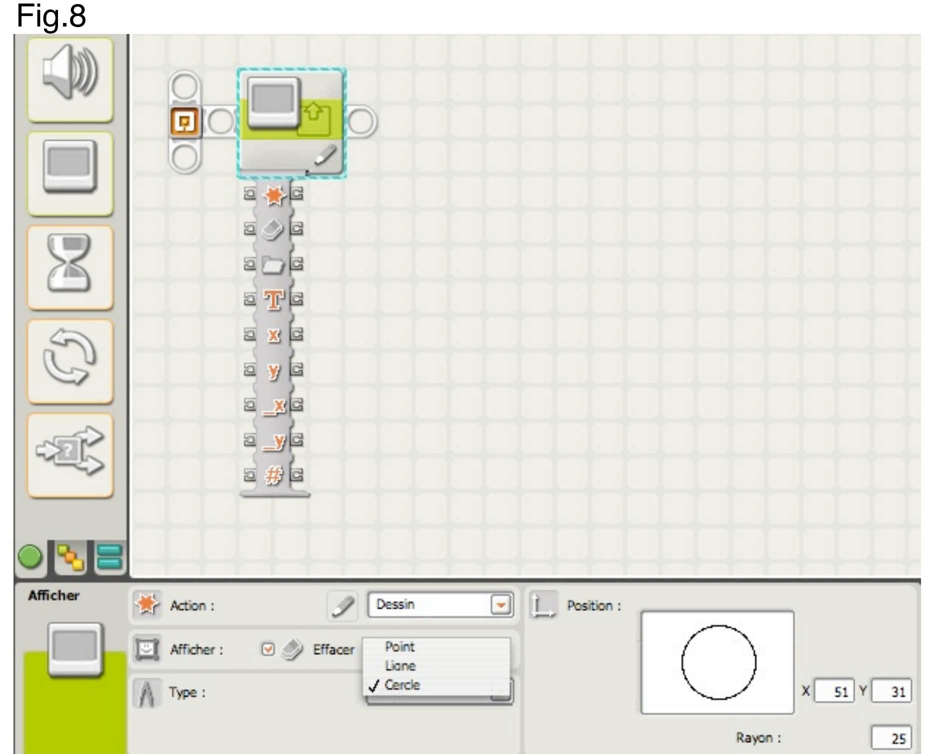
L'écran LCD a une résolution horizontale de 100 pixels et une résolution verticale de 64 pixels. Cela signifie qu'on peut placer 100 petite points horizontaux sur chaque ligne, et 64 verticaux sur chaque colonne à l'intérieur de l'écran. Cela fait donc au total 6400 points. Lorsque tous ces points sont placés, la totalité de l'écran est noir.
Pour positionner ces points, on adopte une convention qui consiste à choisir deux axes de coordonnées perpendiculaires: axe des x et axe des y. Par définition l'origine de positionnement se trouve en haut et à gauche de l'écran.
Les valeurs partent de zéro (0)et croissent vers la droite jusqu'à 99 pour les x; de zéro (0) et descendent jusqu'à 63 pour les y.
Un point quelconque est donc caractérisé par sa position en x et sa position en y. Par exemple, un point 20,12 se situe sur la 21ème ligne et la 13ème colonne (ne pas oublier de compter les zéros).

Fig.7



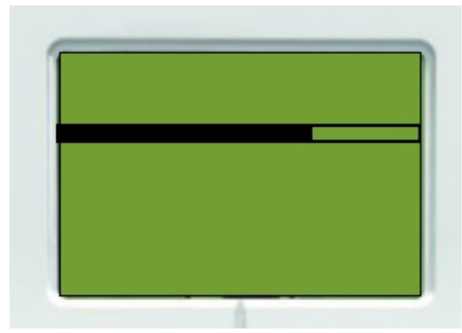
En résumé, la valeur en x/y d'un pixel situé en haut et à l'extrême gauche est 0/0. Sur la même ligne et à l'extrême droite sa valeur est 0/99. Souvenez-vous que la valeur y croît au fur et à mesure que l'on se déplace vers le bas de l'écran; la valeur du pixel situé tout en bas et à gauche est 0/63; elle est 99/63 pour un pixel situé tout en bas et à l'extrême droite.

Revenons à la programmation NXT-G et examinons le bloc AFFICHER.
Fig.8



Dans la série des plots de données, ceux qui nous intéressent plus particulièrement portent l'indication x et y, ainsi que _x et _y. Par ces plots on contrôle la position d'un objet graphique affiché sur le petit l'écran.
Le panneau de configuration est paramétré pour l'affichage d'un cercle de rayon 25 (pixels). Sa position dans la fenêtre de l'écran est fixée par ses coordonnées x = 51 et y = 31.
Si aucun fil de données est raccordé aux plots x et y, l'objet sera vu tel quel. Par contre si les plots x et y sont reliés en entrées à un autre bloc fournissant des valeurs variables, l'objet sera vu dans la position définie par ces valeurs variables. Et si ces variables suivent une loi mathématique, l'objet se déplacera selon cette loi. Et pour donner l'illusion du déplacement, il suffira de créer une boucle contenant le bloc AFFICHER et relier les plots x et y à un 'pourvoyeur' de données, comme par exemple une distance, un niveau de luminosité, une valeur aléatoire, etc..

EXEMPLE:
Reliez la brique NXT à l'aide d'un câble de connexion, à un capteur US sur le port 4.
Nous souhaitons afficher une glissière mobile proportionnelle à la distance mesurée par le capteur. Pour les besoins de cet exemple, nous limiterons la distance capteur-objet à 100 cm.
Le résultat recherché est le suivant:
Fig.9



Nous tracerons sur l'écran 2 lignes parallèles fixes, à l'intérieur desquelles se déplacera une barre noire mobile à partir du bord gauche qui est l'origine. La longueur de cette barre sera proportionnelle à la distance mesurée par le capteur.
Le programme qui en découle ressemble à ceci:

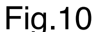
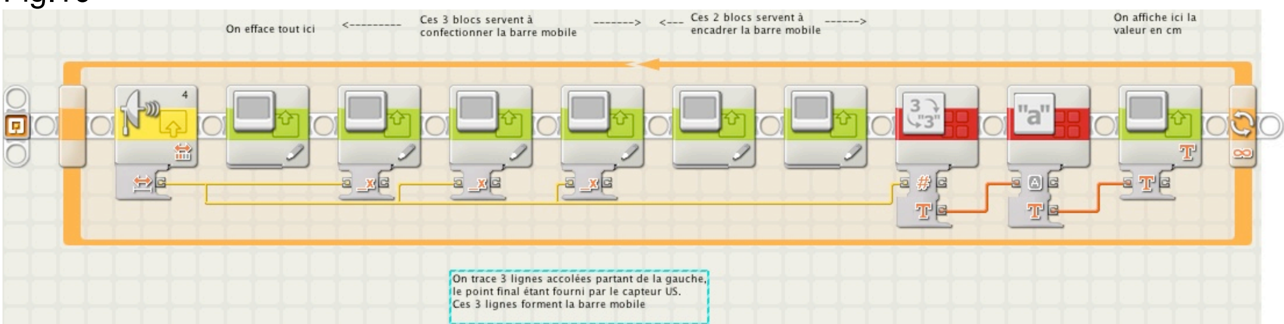


Fig.10

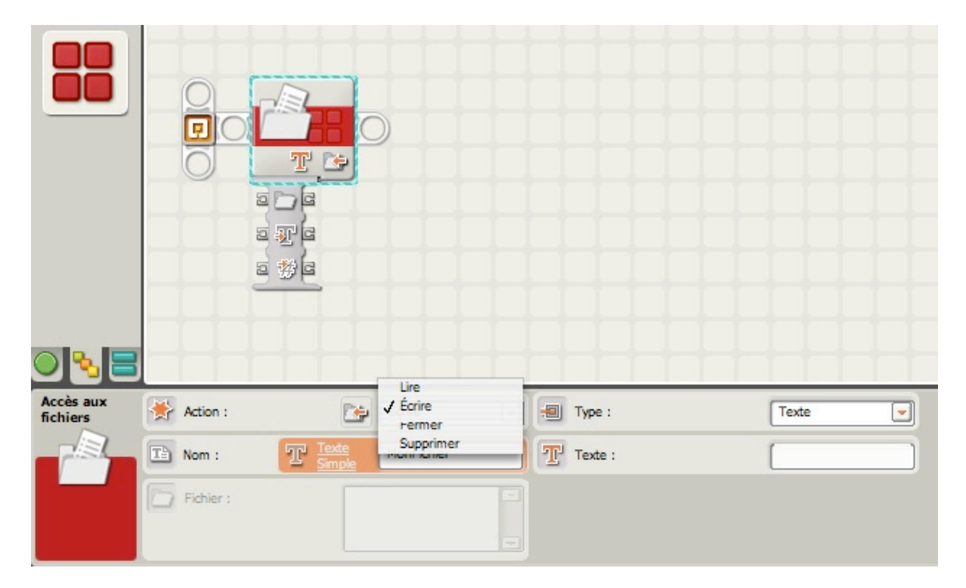


N'importe quel capteur peut traduire de cette manière des valeurs mesurées. Il en est de même pour les servomoteurs pour lesquels on pourrait traduire les puissances et même le sens de rotation, si l'origine se situe non plus au bord gauche, mais dans l'axe vertical de l'écran. Le sens positif ou négatif par rapport à cet axe indique alors le sens de rotation du moteur.
Vous remarquerez cependant le nombre important de blocs AFFICHER nécessaires pour réaliser cette représentation graphique. A noter également que seul le premier bloc AFFICHER est coché pour effacer l'écran à chaque passage de la boucle.

STOCKAGE PERMANENT DE DONNEES dans la mémoire du NXT.

Il est parfois utile, dans certains cas, de conserver des valeurs afin de les retrouver pour une autre utilisation, soit dans un autre programme, soit en récupération pour un traitement suivant.
Pour cela, il existe dans la palette entière "avancé" du logiciel NXT-G, un bloc intitulé "accès aux fichiers".

Fig.11



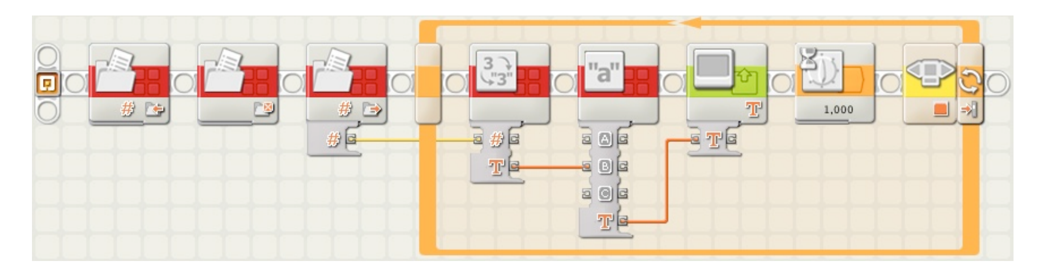
Ce bloc permet d'enregistrer les données du robot dans des fichiers sur la brique NXT.

Dans la zone "Action", nous avons à disposition 4 modes d'utilisation possible.
Dans la zone "Type", les données pouvant être stockées sont de 2 types: soit 'Texte' soit 'Numérique'. Les données logique n'y figurent pas. Mais en choisissant une valeur égale à 0 ou 1, il est possible de les transformer (après récupération) en valeurs logiques.
Dans la zone "Nom" vous choisissez un nom caractéristique qui vous permet de retrouver facilement vos données stockées.
Enfin la zone de saisie contient le texte ou la valeur à enregistrer dans le fichier que vous avez nommé. Ces données peuvent également provenir d'autres blocs (notamment des blocs VARIABLE) à l'aide de fils de données.

Après avoir ECRIT des données dans un fichier, il faut utiliser un autre bloc "Accès aux fichiers" pour FERMER le fichier avant de pouvoir le LIRE ou le SUPPRIMER (à l'aide d'un troisième bloc "Accès aux fichiers").
Par exemple, pour écrire dans un fichier puis lire des données à partir de celui-ci, le fichier doit être FERME entre les deux actions. Cette tâche exigerait trois blocs "Accès aux fichiers" : le premier (pour lequel l'option « Écrire » est activée dans le panneau de configuration) écrit des données dans un fichier ; le deuxième, placé plus loin dans le programme, doit être configuré pour fermer ce fichier ; enfin, le troisième (pour lequel l'option « Lire » est activée dans le panneau de configuration) lit le fichier. Ces trois blocs peuvent être placés côte à côte ou être disposés à différents endroits du programme.
Enfin l'écriture dans un fichier existant AJOUTE les données à la fin du fichier, au lieu d'effacer les données existantes. Pour ECRASER un fichier, il faut tout d'abord utiliser un bloc "Accès aux fichiers" qui SUPPRIME le fichier, puis en employer un second qui ECRIT un nouveau fichier en lui donnant le même nom.

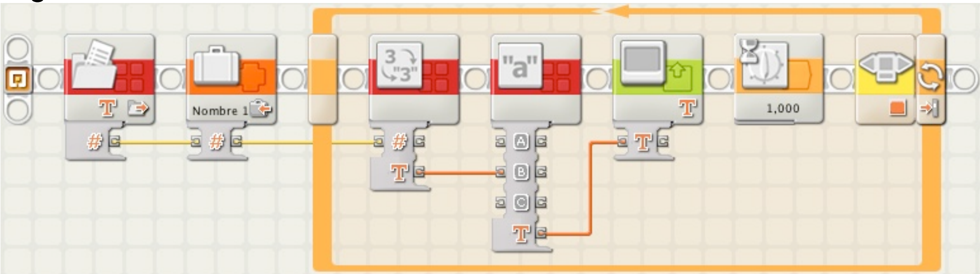
Voici à quoi pourrait ressembler un programme d'enregistrement de données:

Fig.12



Et voici comment procéder pour récupérer ces valeurs, à l'aide d'un autre programme:

Fig.13



PROGRAMMES UTILES

Et maintenant, vous trouverez ci-jointe une série de programmes que vous pourrez transformer à votre guise pour vos propres robots. Certains vous paraîtront simplistes, mais à l'usage ils vous éviteront des recherches inutiles. D'autres, basés sur un concept différent produisent des effets identiques: c'est une façon de traiter un même problème de plusieurs manières.
Tous ces programmes peuvent s'intégrer dans des ensembles plus complexes.
Ayez toujours en tête la simplification, et n'hésitez pas de tester des blocs qui vous paraissent inutiles. Ils sont parfois la réponse à une question sans réponse.
Enfin, je vous invite à enrichir cette liste par vos propres contributions. Si vous pensez avoir trouvé une astuce ou un moyen de simplification, n'hésitez pas à les proposer.

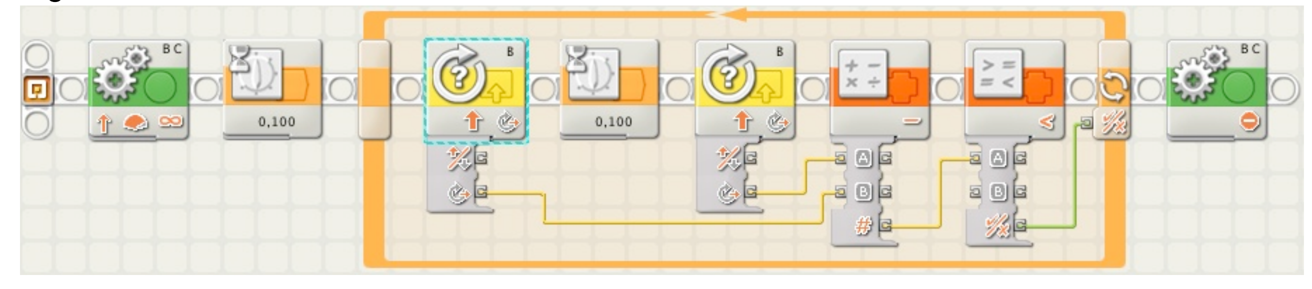
Quelques exemples:

Contrôle direct des moteurs
Marche et arrêt

Fig.1

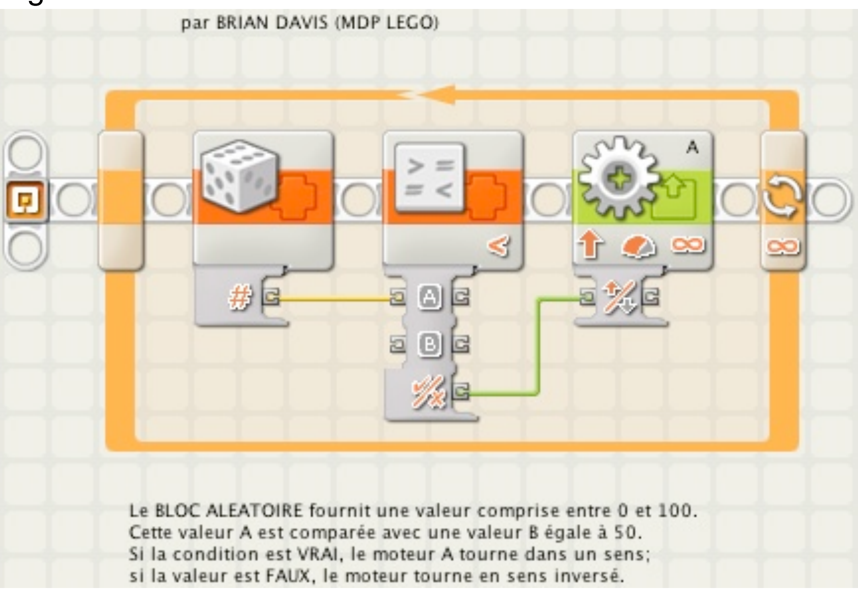


Fig.2



Déplacements linéaires et virages.

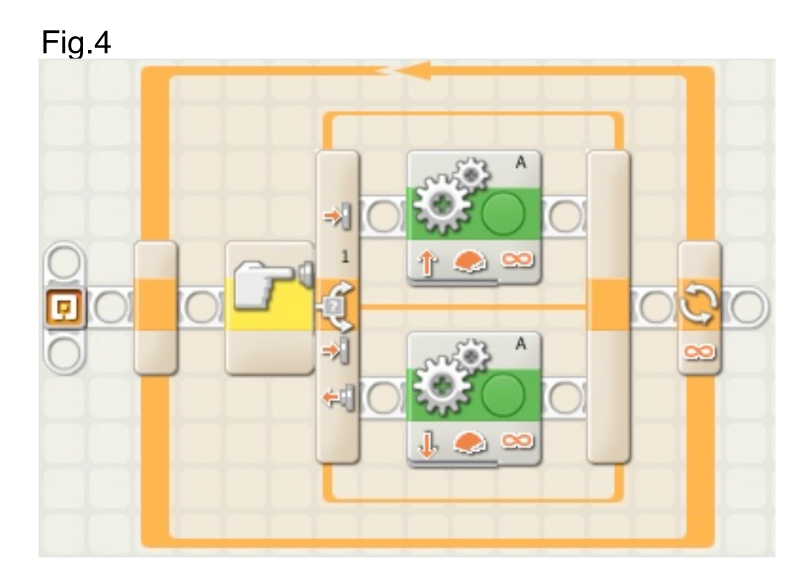
Fig.3



Contrôle par capteurs

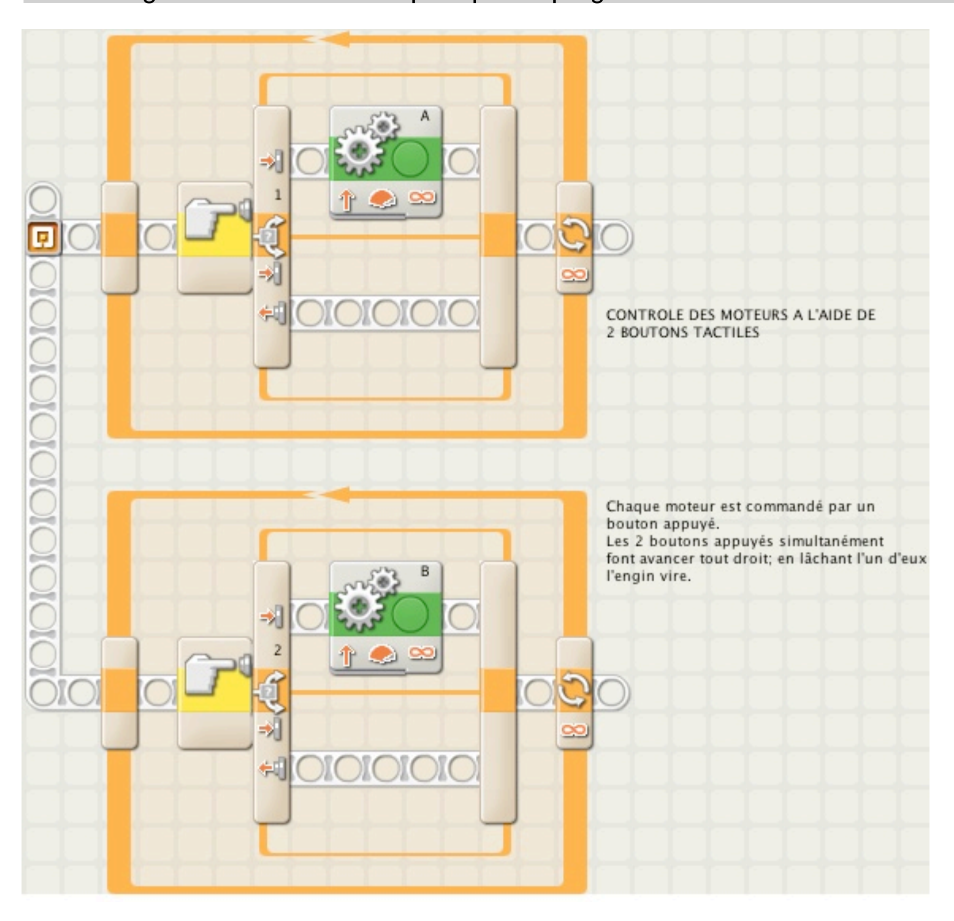
Inversion de marche par impulsion sur le capteur tactile

Fig.4



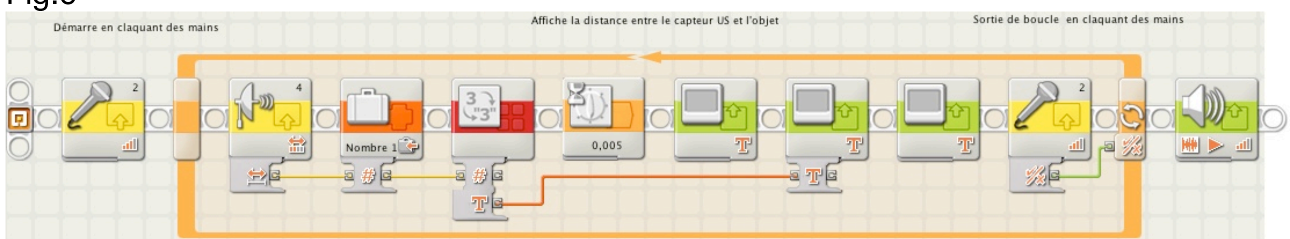
Contrôle des moteurs par 2 capteurs tactiles

Fig.5



Démarrage ou arrêt par le son

Fig.6

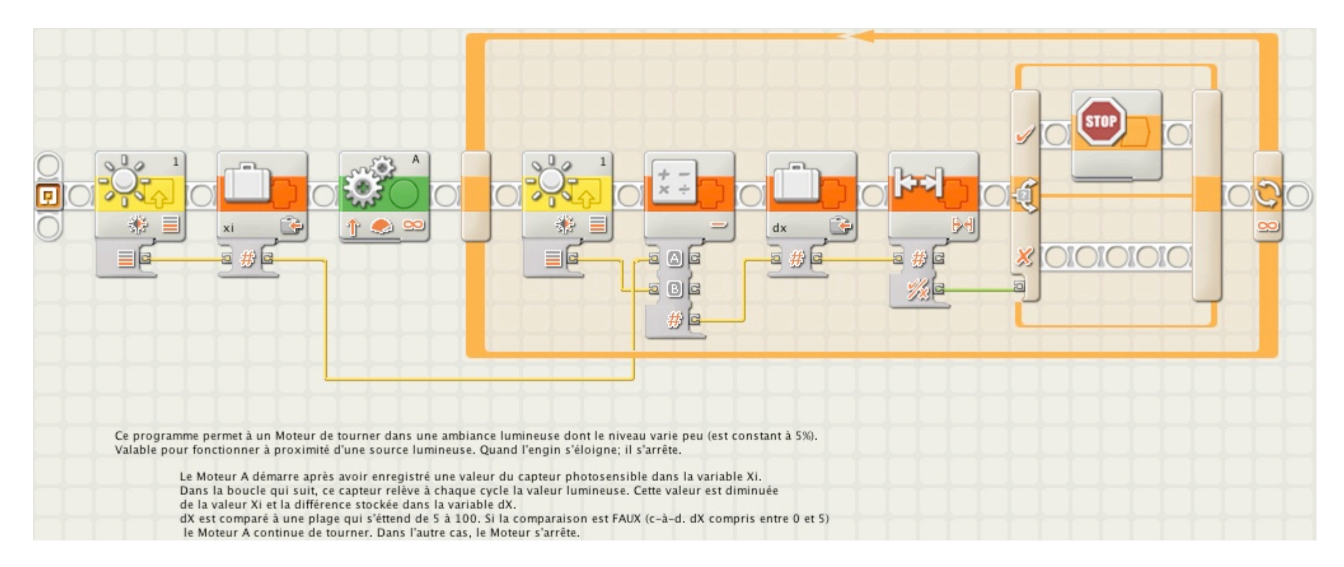


Ce programme démarre en tapant des mains. Le capteur à Ultrasons affiche en cm la distance qui le sépare d'un objet. La sortie de boucle se fait également en tapant des mains.

Déplacements particuliers:

Dans une direction

Fig.7



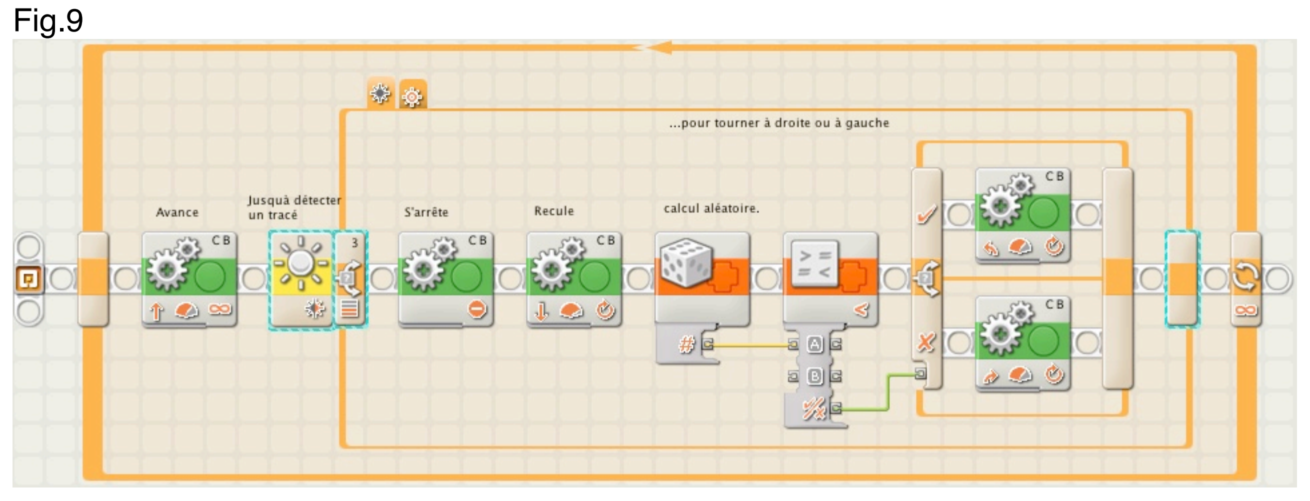
Suivre un tracé

Fig.8



A l'intérieur d'un tracé

Fig.9



Le robot se déplace à l'intérieur d'une forme sans en sortir.



Leçon n°14: UN CONTROLEUR PID pour ROBOTS...

Nous allons aborder ici un aspect particulier du contrôle des robots, notamment la relation entre capteurs et moteurs, c'est-à-dire la manière dont ils interagissent les uns sur les autres.

PID désigne un dispositif, un organe de contrôle informatisé utilisé dans l'industrie pour une grande variété de machines incluant les véhicules, les robots et même les engins balistiques.
Donner ici une définition mathématique du PID est assez difficile pour ceux qui ne disposent pas de connaissances suffisantes et ne maîtrisent pas les calculs, mais une bonne compréhension du dispositif est nécessaire pour utiliser efficacement un PID.

Cette leçon décrit la façon de créer un PID destiné aux robots Lego Mindstorms NXT programmés en NXT-G version 1.1..
Elle est inspirée d'un article rédigé par James Sluka et publié le 29 septembre 2009 sur le blog NXT STEP
http://thenxtstep.blogspot.com/2009_09_01_archive.html

CHAPITRE 1:

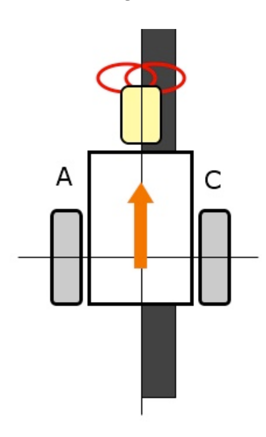
Le plus simple est de prendre un exemple, et dans notre cas, ce sera un Robot suiveur d'un tracé.
Une fois créé, le même PID peut-être employé moyennant quelques modifications mineures, pour n'importe quelle autre action, comme se diriger d'une manière rectiligne, ou grimper une rampe, ou encore se balancer en utilisant seulement 2 roues en contact avec le sol.

Pour éviter de se lancer dans des notions trop compliquées de calculs, et pour que cette approche soit à la portée de tous, je partirai d'un concept simple faisant appel aux mathématiques d'une manière très limitée.
UN PID est un outil assez simple, et sa description est facilement comprise par tous ceux qui calculent sans trop de difficultés.

Eléments de base

Commençons par fixer les éléments de base d'un robot suiveur de tracé.

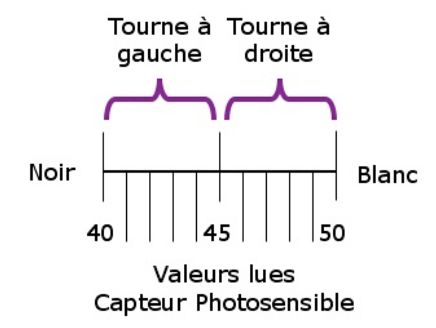
fig.1



Ce robot schématiquement représenté, dispose de 2 roues arrières A et B entraînées chacune par un moteur.
 à l'avant une roue folle. Ce dispositif permet une direction différentielle en jouant sur la vitesse des 2 moteurs. Egalement à l'avant, un capteur photosensible pointant vers le bas, de telle sorte qu'il ne "voit" que le support sur lequel il est posé. La flèche indique le sens de déplacement du robot.
L'objectif recherché est de contraindre le robot à suivre un tracé matérialisé sur le support par une large bande noire.
Ce robot, "suiveur de tracé" peut-être construit avec un capteur photosensible ou plusieurs si vous en disposez. Plus vous installerez de capteurs et meilleure sera cette conduite. Dans notre cas nous nous limiterons à un seul capteur, car même équipé de la sorte, notre robot pourra suivre le tracé avec précision, quelle que soit sa forme, courbes comprises. La seule chose que vous perdrez, c'est la vitesse de déplacement.

La première astuce que nous utiliserons, et qui est sans rapport avec un PID, sera de suivre non pas le tracé, mais le bord du tracé. Pourquoi?
Parce que, si nous suivons le tracé lui-même (bande noire) alors que le robot s'écarte de la ligne et que le capteur "voit blanc", nous ne savons pas de quel côté du tracé nous nous trouvons. Sommes nous à droite ou à gauche? Mais si nous suivons le bord du tracé, alors nous pouvons savoir de quel côté nous nous trouvons quand le robot dévie. Si le capteur "voit blanc" alors nous savons qu'il s'agit du bord gauche. Et s'il "voit noir" nous savons qu'il se trouve à droite du bord. On appelle cela un "suiveur de ligne gauche" (*left hand line follower*).
Nous avons maintenant besoin de connaître les valeurs fournies par le capteur quand il "voit blanc" et quand il "voit noir". D'ordinaire, un Capteur Photosensible (non calibré) donne une valeur de 50 pour un "blanc" et 40 pour un "noir" (échelle de 0 à 100).
Si on représente ces valeurs sur une ligne graduée, on pourrait résumer les lectures de la manière suivante:

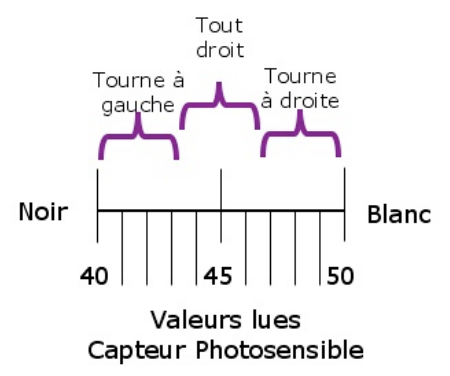
fig.2



Bien, on a donc coupé la plage de valeurs en 2 tranches égales, et on peut dire que si la valeur relevée est inférieure à 45, le robot tournera à gauche, et si elle est supérieure à 45, il tournera à droite.
Pour l'instant la manière de virer ne sera pas abordée. Disons que sur une ligne plutôt rectiligne les virages seront mieux maîtrisés. Pour un tracé présentant de nombreuses courbes, il faudra prévoir des virages plus serrés. Dans le cas d'une trajectoire rectiligne, on prévoira un niveau de puissance égal à 50% pour la roue rapide, et 20% pour la roue lente. Pour des virages plus serrés sur une trajectoire courbe, on choisira un niveau de 30% pour la roue rapide et un freinage pour la roue lente. Quels que soient les niveaux de puissance, il seront identiques pour les 2 virages (droite et gauche); il faudra seulement inverser les valeurs des 2 moteurs.
Cette manière de suivre un tracé manque d'élégance. Elle peut paraître suffisante pour des trajectoires à peu près rectilignes, mais quand il s'agit de trajectoires sinueuses, il faut demander au robot de "serpenter" le long du tracé puisqu'il ne sait faire que 2 choses: tourner à gauche ou à droite. De plus, ce dispositif ralenti le temps de parcours et offre un spectacle désolant.

Dans cette approche le robot ne suit jamais une trajectoire rectiligne, même s'il est parfaitement aligné sur le bord du tracé. Cela ne paraît pas très efficient, n'est-ce pas?

Procédons autrement. Au lieu de diviser la ligne graduée (fig.2) en 2 parties, divisons la en 3.
fig.3



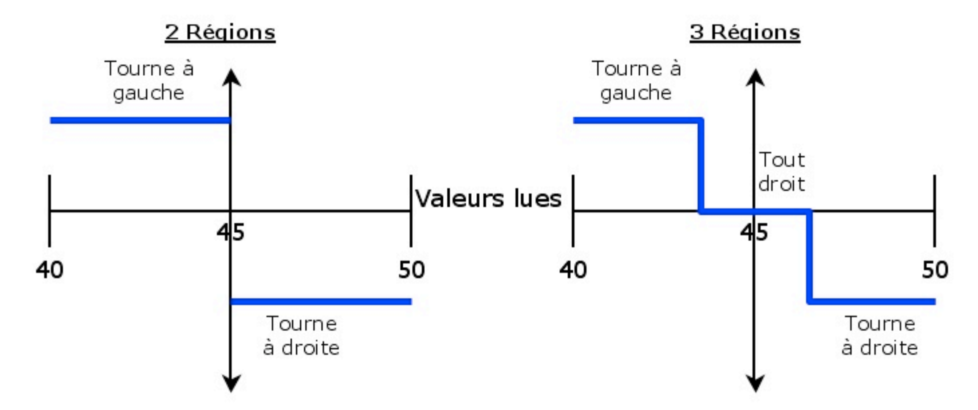
A présent, si le niveau lumineux est inférieur à 43, nous voulons que le robot tourne à gauche. Si le niveau est entre 44 et 47, le robot doit aller tout droit. Enfin si le niveau est supérieur à 47, le robot doit tourner à droite. Cette relation peut-être facilement établie en programmation NXT-G à l'aide de 2 commutateurs imbriqués (2 tests à faire et non 3).
Cette approche fonctionne mieux que la précédente. Nous savons maintenant que le robot peut avancer quelquefois en ligne droite.

Si le partage de la ligne des valeurs en 3 régions donne de meilleurs résultats qu' en 2 régions, qu'en est-il si on augmente ce nombre découpage?
C'est ici qu'on aborde le PID.

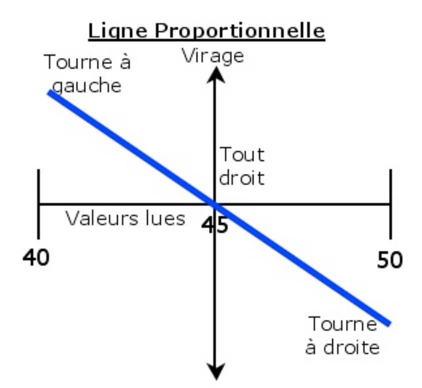
1 - Le "P" dans PID: P pour Proportionnel

Que se passe-t-il donc si on augmente le nombre de divisions dans la ligne de valeurs?
Il faut d'abord s'interroger sur le sens à donner au mot "Tourne" lorsque le nombre de régions dépasse 3. Dans la première approche, le robot ne pouvait faire que 2 choses; tourner à droite ou tourner à gauche. Les virages sont strictement identiques, mais de

sens opposé. Dans la 2ème approche, on a ajouté "tout droit" aux deux précédents. Mais si nous dépassons 3 régions, il nous faut plus de "sortes" de 'Tourne', je dirai de Virages.
Fig.4



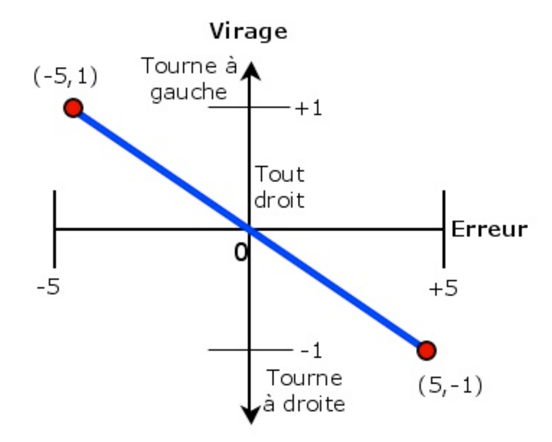
Dans le cas suivant, nous sommes en présence de ce que nous appellerons une ligne Proportionnelle. C'est à quoi ressemblerait la ligne bleu en escalier, lorsque le nombre de régions (gradins) augmenterait indéfiniment. Dans ce cas, les virages se produisent avec douceur entre ces deux limites. Si la lecture du capteur dit que l'on est proche de la ligne, alors on exécute un petit virage. Si l'on se trouve à une plus grande distance de cette ligne, alors le virage sera plus important.
Fig.5



***Proportionnel** signifie qu'il existe une relation linéaire entre 2 variables.*

Pour comprendre ces "plus de sortes"de 'Tourne', nous allons utiliser une représentation graphique dans un système d'axes en X et Y. L'axe horizontal (axe des X) supporte les valeurs lues du niveau de luminosité tel qu'il est représenté. L'axe Vertical (Axe des Y) sera celui des Virages.
Pour simplifier, nous dirons que *proportionnel* signifie qu'il existe une relation entre ces 2 variables, dont la représentation graphique est une ligne droite.
Vous devez sans doute savoir que l'équation algébrique d'une ligne droite se présente sous la forme :
$y = mx + b$
Où y est la valeur verticale (positive ou négative) sur l'axe des Y, et x la valeur correspondante sur l'axe des X. La constante m est la pente de cette droite et b est la valeur de Y quand cette droite coupe l'axe des Y, c'est-à-dire quand x est égal à zéro.
La pente se définit comme une valeur résultant de la division de l'écart y (variation de y) par l'écart x (variation de x) correspondante, de 2 points rapprochés pris sur cette droite.
Si vous ne savez pas grand choses sur les droites (ou si vous avez oublié), je ferai donc un petit rappel en simplifiant le graphique et l'équation. D'abord, nous remettrons le centre de notre droite de valeurs lues (l'Axe des X) au zéro. C'est facile à faire. Aux valeurs

extrêmes 40 et 50 du niveau lumineux, nous soustrairons juste 45 (qui est la moyenne de 40 et 50, soit (40+50)/2) de toutes les valeurs relevées. Nous appellerons ce résultat l'***erreur***. Ainsi, si la valeur est 47 nous soustrairons 45 et obtiendrons une erreur = 2. L'**erreur** nous donne la valeur séparant notre position jusqu'au bord du tracé . Si le capteur est exactement sur la ligne notre **erreur** est égale à zéro puisque nous soustrayons 45 de toutes nos lectures. Si le capteur est entièrement dans le blanc notre **erreur** est égale à + 5, et entièrement dans le noir égale à - 5.
Fig.6



Dans ce graphique, on a remplacé les valeurs de niveau lumineux de l'axe des X par une nouvelle échelle qui est celle des **erreurs**. Et comme cette droite coupe l'axe des Y à l'origine c'est-à-dire au point zéro; l'équation se simplifie et devient:
y = mx
Et si on adopte une expression plus générale, on peut écrire:

Virage = m * **erreur**
(Dans ce qui suit, il faut comprendre le mot «virage» comme correction de la trajectoire).

Mais, nous n'avons pas encore défini la signification de l'axe des virages. Aussi pour l'instant contentons-nous de dire que la plage des valeurs se situe entre -1 (virage franc à droite) et +1 (virage franc à gauche); le zéro signifiant tout droit. La pente de cette droite peut-être calculée en utilisant les coordonnées des 2 points rouge situés aux extrémités de la droite:

pente = m = (écart des y) / (écart des x) = (1 - (-1)) / (-5 -(5) = - 2/10 = - 0,2 soit 0,20 en valeur absolue.

La pente est une **constante proportionnelle** qui est le facteur multiplicateur à appliquer à l'**erreur** (valeur de x) pour obtenir celle du **virage** (valeur de y).
Dans le vocabulaire adopté pour exprimer le PID, cette constante est appelée " **K** " (peut-être pour rappeler la notion de constante). On peut l'assimiler à un *facteur de conversion* qui, à partir d'un niveau de luminosité ou d'une erreur dans notre cas, produit une autre valeur comme celle d'un virage. Très simple et très puissant!

Donc, en utilisant ces nouvelles conventions, on peut écrire l'équation de cette droite ainsi:

Virage = K * (erreur)

La valeur **Virage** est considérée comme une sortie de notre Contrôleur **P**, et désignée par le terme P.

Vous avez remarqué que dans le dernier graphique, la ligne erreur est dans la plage -5 et +5. En dehors de cette plage, il n'est plus possible de déterminer l'éloignement du capteur par rapport au tracé. Tous les "blanc" sont identiques si le capteur n'est pas en mesure de détecter un "noir". Souvenez-vous que cette plage est arbitraire, car elle est déterminée par les caractéristiques du capteur photosensible., les couleurs des supports, etc…. Dès que le capteur s'éloigne trop du tracé, il relève des valeurs constantes ; cela signifie que les lectures ne sont plus proportionnelles à l'erreur. La déviation par rapport au tracé ne peut être appréciée que si le capteur est assez proche de ce tracé. C'est à l'intérieur de cette plage que les valeurs lues sont proportionnelles à la distance. A l'extérieur de cette plage, il nous indique la bonne direction, mais l'ampleur est faussée. La lecture des valeurs, ou l'erreur, est plus petite qu'elle ne l'est en réalité, et ne fournit donc pas une réponse satisfaisante.

Dans la définition du PID, la plage qui nous intéresse et qui donne une réponse proportionnelle est appelée "**Plage Proportionnelle**". Ce concept est très important dans la notion du PID.
Pour revenir à notre suivi de tracé, la **Plage Proportionnelle** pour le Capteur Photosensible se situe entre 40 et 50, et pour **l'erreur** elle est comprise entre -5 et +5. On peut également noter que les servomoteurs ont eux aussi une plage proportionnelle, de -100 (pleine puissance arrière) à +100 (pleine puissance avant).

Deux remarques importantes sur la Plage Proportionnelle:
(1) Vous souhaitez une plage proportionnelle aussi grande que possible.
La plage proportionnelle du Capteur Photosensible est assez petite, cela signifie que le Capteur doit être assez près du bord du tracé pour obtenir des valeurs proportionnelles. Plus exactement, la grandeur de la plage dépend essentiellement de la hauteur du capteur par rapport au plan du tracé. Si le capteur est très proche du plan, disons 2 mm environ, il émet un très petit cercle lumineux. Un petit déplacement du capteur produira une variation de **l'erreur** de -5 à +5, correspondant à notre plage proportionnelle. Vous pourriez dire que le capteur a "des vues étroites" et il peut seulement voir une très petite partie du plan. Le capteur doit être très près du bord du tracé pour obtenir une lecture qui n'est pas "blanc" ou "noir". Si le capteur est positionné plus haut, la tache lumineuse s'étale en un plus grand cercle sur le plan. À une hauteur de 12 mm environ, le capteur émet une tache lumineuse de même dimension.   A cette hauteur, la plage proportionnelle est trop grande, le capteur devant se maintenir entre +/- 12 mm de part et d'autre du bord du tracé pour obtenir un résultat proportionnel satisfaisant.
Malheureusement, il y a deux inconvénient pour une position trop élevée du capteur. D'abord le capteur "voit" et réagit à l'éclairage ambiant beaucoup plus que s'il se trouve en position basse. Ensuite, il discerne moins bien les nuances entre le noir et le blanc. A une certaine distance, il donnera une même lecture au noir et au blanc.

(2) À l'extérieur de la plage proportionnelle le contrôleur provoquera un déplacement dans une direction correcte mais sous-évaluée. La réponse proportionnelle du contrôleur est limitée par la plage proportionnelle.

2 - Virage et niveau de puissance réelle du moteur.

Comment pouvons-nous mettre en oeuvre les virages ? Quel doit être le niveau de puissance des moteurs ?

Une façon d'envisager les virages consiste à définir "un Niveau de Puissance Cible", que nous appellerons "**Tp**". **Tp** est le niveau de puissance des deux moteurs quand on suppose que le robot se déplace tout droit, ce qu'il fait quand **l'erreur** = 0.
Quand **l'erreur** n'est pas nulle nous utilisons l'équation **virage = K * (l'erreur)** pour calculer le changement des niveaux de puissance des deux moteurs. Un moteur obtiendra un niveau de puissance égal **Tp+Virage**, l'autre moteur obtiendra un niveau de puissance égal à **Tp-Virage**.
A noter que notre **erreur** est comprise entre -5 à +5, ce qui signifie que le **Virage** peut être positif ou négatif c'est-à-dire, correspondant aux virages dans des sens opposés. Il se trouve que c'est exactement ce que nous recherchons puisqu'il mettra automatiquement un moteur en accélération et l'autre en ralentissement.
Un moteur (nous supposerons que c'est le moteur gauche du robot raccordé au port A) obtiendra toujours la valeur de **Tp+Virage** comme niveau de puissance. L'autre moteur (le droit raccordé au port C) obtiendra toujours la **Tp-Virage** comme c'est le niveau de puissance. Si **l'erreur** est positive, alors **Virage** est positif et **Tp+Virage** est plus grand que Tp; la vitesse du moteur gauche augmentera alors que celle du moteur droit diminuera.
Si **l'erreur** change de signe et devient négative (signifiant qu'on a franchi le bord du tracé et voit l'autre couleur), alors **Tp+Virage** est maintenant plus petit que **Tp**, la vitesse du moteur gauche augmente et celle du moteur droit diminue, puisque **Tp-Virage** est plus grand que **Tp** (rappelez-vous que le produit de 2 valeurs négatives donne une valeur positive). Simple, non?
Poursuivons.

3 - Pseudo Code pour un Contrôleur P

D'abord nous devons évaluer les valeurs lues par le capteur de lumière pour le blanc et le noir. De ces deux valeurs nous pouvons calculer la moyenne, c'est-à-dire quelle valeur soustraire à celle d'une lumière brute lue pour convertir l'ensemble en valeur d'erreur. La moyenne est la demi-somme des lectures "blanc" et "noir". Pour la simplicité on supposera que cette moyenne a déjà été mesurée et stockée dans une variable appelée *moyenne*. (Une correction intéressante consisterait à faire calculer la *moyenne* par le robot à partir des mesures lues niveaux blanc et noir).

Nous aurons aussi besoin d'un emplacement de stockage pour la constante K, nous l'appellerons **Kp** (de constant pour le contrôleur proportionnel). Ainsi qu'une valeur de départ pour **Kp**.
Il y a plusieurs façons pour l'obtenir, par tâtonnement puis corrections successives de l'erreur. Ou alors, vous pouvez essayer d'évaluer une valeur basée sur les caractéristiques du capteur et du robot. Nous appliquerons cette dernière méthode.
Nous utiliserons un **Tp** (niveau de puissance cible) de 50: quand l'erreur est égale à zéro les deux moteurs fonctionneront au niveau 50. La plage d'erreur se situe entre -5 à +5. Nous supposerons que la puissance variera de 50 à 0 quand l'erreur passera de 0 à -5. Cela signifie que **Kp** (la pente souvenez-vous, la variation de y divisée par la variation de x correspondante) est:

Kp = (0 - 50) / (-5 -0) = 10

Nous utiliserons la valeur 10 de Kp, pour convertir une valeur **erreur** en valeur **Virage.** En d'autres termes: une variation de 1 de l'erreur, se traduira par une augmentation de 10 de la puissance d'un moteur. L'autre moteur verra sa puissance diminuer de 10.

Aussi, en pseudo-code, c'est-à-dire, sous forme d'énumérations logiques d'actions, indépendamment du langage de programmation utilisé (NXT-G ou autre), nous pourrions écrire:

```
Kp = 10                              ! Initialisation de trois variables
moyenne = 45
Tp = 50
Loop forever                         ! Début de Boucle pour toujours
   LightValue = valeurlueducapteur    ! Quelle est la valeur lue courante du
capteur?
   erreur = LightValue - moyenne      ! calcul de l'erreur par soustraction de
la moyenne.
   Virage = Kp * erreur               ! le terme "P", de combien veut-on
modifier la puissance du moteur
   powerA = Tp + Virage               ! niveau de puissance pour le moteur A
   powerC = Tp - Virage               ! niveau de puissance pour le moteur C
   MOTOR A direction=forward power=powerA !     transmet la commande avec la
nouvelle valeur de puissance dans un bloc MOTEUR.
   MOTOR C direction=forward power=powerC ! Commande identique à l'autre moteur
mais en utilisant l'autre valeur de puissance.
end loop forever                     ! fin de boucle, retour au début de boucle
et recommencer à nouveau.
```

Bien, nous y voilà presque. Il y a cependant un problème subtil qui devrait être corrigé. Mais laissez de toute manière une chance à votre robot. S'il semble s'écarter du bord du tracé, au lieu de s'en rapprocher, la cause la plus probable est que vous avez inversé les sens des Virages. Corrigez Kp à -10 et vérifiez le résultat Si cela rectifie les sens de Virage alors revenez à Kp en conservant +10 et changent les signes dans les deux lignes de la manière suivante;

```
powerA = Tp - Virage
powerC = Tp + Virage
```

Il y a deux "paramètres variables" et une constante dans ce contrôleur **P**. La constante est la valeur moyenne (la moyenne des lectures du capteur photosensible pour le blanc et le noir). Vous devrez donc écrire un court programme pour évaluer les niveaux lumineux sur votre surface d'évolution de votre robot. Vous avez besoin de déterminer une valeur "noir" et une valeur "blanc". Calculez la moyenne et placez-la dans le programme du contrôleur **P** (dans la variable Moyenne).
La plupart des lignes qui vont suivre supposent que vous avez réalisé ce petit programme exécuté par le robot.

La valeur de **Kp** et la puissance cible **Tp** sont des paramètres variables. Un paramètre variable doit être choisi par tâtonnements et approximations successives. **Kp** contrôlera la vitesse du contrôleur dans ses mouvements de retour au bord du tracé quand il s'en est éloigné. **Tp** contrôlera la vitesse du robot dans son déplacement le long du tracé.
Si la ligne est pratiquement rectiligne, vous pouvez utiliser une grande valeur pour **Tp** afin de déplacer le robot à grande vitesse, et une petite valeur pour **Kp** pour rendre les virages plus doux.
Si le tracé présente des courbes serrées, La valeur maximum **Tp** sera la meilleure. Si **Tp** est supérieur à ce maximum, la valeur de **Kp** importe peu, et le robot ratera son virage car il se déplace trop rapidement. Si **Tp** est faible, la plupart des valeurs de **Kp** seront acceptables compte tenu du lent déplacement du robot. L'objectif est d'avoir un déplacement le plus rapide possible sans perdre la capacité de suivre le tracé.

Nous avions supposé une valeur de départ pour **Kp** de 10. Pour **Tp** vous pourriez choisir une valeur plus faible que celle suggérée ci-dessus, peut-être 15 (le robot se déplace assez lentement). Essayez et vérifiez le résultat. Si vous ratez le tracé parce que le robot semble virer trop lentement, alors augmentez **Kp** et essayez à nouveau. Si vous ratez le tracé parce que le robot semble trop réactif dans la recherche du tracé dans les deux sens diminuez alors **Kp**. Si le robot semble suivre le tracé correctement, augmentez **Tp** et assurez-vous de pouvoir suivre le tracé à une vitesse plus rapide. Pour chaque nouvelle valeur de **Tp** vous devrez déterminer une nouvelle valeur correspondante de **Kp**, sachant que **Kp** présente d'habitude peu de variation.

Suivre un tracé rectiligne est d'ordinaire assez facile. Suivre un tracé comportant des courbes douces est un peu plus délicat. Suivre enfin un tracé comportant des courbes serrées est nettement plus dur.
Si le robot se déplace assez lentement, presque n'importe quel tracé peut être suivi, même avec un contrôleur rudimentaire. Nous recherchons un bon suivi de tracé, avec une vitesse convenable et la capacité d'aborder avec succès les virages doux. (Les tracés avec virages serrés suivent généralement des trajectoires de tracés plus spécialisées).

Il est évident que le meilleur contrôleur **P** est spécifique à chaque type de tracé (largeur, rayons des courbes, etc.) et à chaque robot. En d'autres termes, un contrôleur **P** (ou un contrôleur PID en la matière)
est conçu pour une seule sorte de tracé et de robot et ne fonctionnera pas convenablement pour d'autres tracés ou robots. Il faudra donc adapter les paramètres Kp et Tp selon les circonstances.

Avant d'aller plus loin, Je vous propose deux exercices dont l'utilité se retrouvera par la suite.

Premier exercice:
Utiliser le tapis d'évolution (test pad) fourni dans le kit # 8527.
Ecrire un programme NXT-G capable de lire les valeurs fournies par le capteur photosensible pour le 'blanc' et le 'noir' à partir du tracé ovale du tapis d'évolution. Puis calculer la moyenne et la stocker dans une variable (cette *moyenne* permettra de calculer par la suite **l'erreur**).
Pour accomplir ce travail, placer le robot, le nez sur le tracé du tapis.
Au début du programme, le robot doit accomplir un balayage en pivotant autour de la roue A (1/2 rotation du moteur A). Pendant ce balayage, capter les valeurs et les stocker dans des variables. Arrêt du robot. Une fois ces valeurs captées, faire revenir le nez du robot à proximité du tracé. Arrêt du robot et affichage de la valeur moyenne sur le petit écran.

Deuxième exercice:
Transformer le programme précédent en un 'Monbloc' en utilisant la palette 'Perso.' Ce 'Monbloc' est une routine qui sera utilisée pour l'écriture du programme futur du contrôleur **P**. Ce bloc est aussi très utile pour l'écriture d'autres programmes faisant appel aux valeurs lues du capteur.

CHAPITRE 2: Un programme simple d’un contrôleur ‘P’ID

Nous utiliserons ici un bloc 'DEPLACER'.

Pour écrire ce programme, nous avons besoin d'une routine ('dans la palette Mon Bloc) que nous appellerons:
PID_LF_Calib.
Cette routine permet de calibrer le capteur par lecture des valeurs lues mini et maxi. Elles s'obtiennent par balayage de la zone où se trouve le tracé. De ces valeurs, la routine en tire une moyenne appellée 'gray'.

Cette valeur est alors reprise par le programme que nous allons à présent développer.

La valeur 'gray' est transmise aux moteurs A et C par l'intermédiaire d'un bloc 'DEPLACER'. A noter que plus le niveau de puissance est élevé, plus le robot aura de difficultés à suivre le tracé.
Ce contrôleur est un suiveur gauche de tracé (left-hand edge follower), qui active le moteur A à gauche et le moteur C à droite. Le capteur photosensible est raccordé au port 2.

La constante proportionnelle Kp est multipliée par 100 pour tenir compte de la nature des nombres exprimés en valeurs entières (ce qui exclus la notion de virgule flottante).

La meilleure valeur pour Kp dépends de plusieurs facteurs: les roues du robot, la plage des valeurs et la hauteur du capteur par rapport à la surface d'évolution. Le programme fonctionnera correctement si le capteur se trouve entre 6 et 12 mm environ au dessus du plan, et si l'éclairage du local est uniformément réparti.

Mais, auparavant il est nécessaire de préciser un point particulier: celui de la virgule décimale.
Il faut rappeler ici que la version NXT-G 1.1 ne traite que des valeurs numériques entières , alors que la version 2.0 prend en compte les valeurs décimales. Cette différence est cause d'une petite complication.

Dans le processus de réglage du contrôleur **P**, la valeur de **Kp** ne cesse de varier entre des limites. La gamme attendue des valeurs que **Kp** dépend très exactement de l'action du contrôleur **P**. Quelle est la dimension de la plage d'entrée et celle de la plage de sortie ?
Pour notre contrôleur **P** (de suivi de tracé), la plage d'entrée est environ 5 unités et la plage de sortie de la puissance des moteurs est de 100 unités , donc il semble probable que **Kp** sera aux alentours de 100/5=20.
Dans certains cas **Kp** attendu ne sera pas aussi grand. Et qu'arrive-t-il si le **Kp** attendu est égal à 1 ?
Puisque les variables dans NXT-G sont limitées aux entiers, quand il s'agit de déterminer les valeurs de **Kp** tout ce qu'il est possible d'entrer c'est...-2,-1, 0, 1, 2, 3. Vous ne pouvez pas entrer 1.3 donc vous ne pouvez pas essayer **Kp** = 1.3. Vous ne pouvez pas utiliser un nombre avec un virgule décimale!
Mais il y aura probablement une grande différence dans le comportement de robot quand **Kp** passera brutalement de 1 à 2. Avec Kp = 2 le robot tentera de corriger l'erreur deux fois plus durement comparée à Kp = 1. La puissance du moteur variera de plus du double pour un même changement de niveaux lumineux. Trop brutal! Ce que nous recherchons, c'est un mode de contrôle plus fin pour **Kp**.
Ce problème est facile à résoudre. Tout ce que nous ferons, c'est de multiplier **Kp** par dix pour redéfinr la plage utilisable dans la limite des entiers. Si on s'attend à ce que **Kp** puisse être proche de 1 alors une valeur de 100 comme multiplicateur serait un bon pari. En effet, il est probablement plus judicieux de toujours utiliser **100*Kp** comme nombre à

entrer dans le programme. Une fois **Kp** multiplié par 100 nous pouvons maintenant entrer ce qui aurait été 1.3 comme 130. 130 n'a aucun point décimal donc NXT-G y trouve son compte.

Mais cela ne complique-t-il pas le calcul ? Oui, mais il est facile de corriger. Une fois que nous avons calculé le terme de **P** nous le diviserons par 100 pour faire disparaître notre multiplicateur. Rappelez-vous notre équation qui définit le contrôleur **P**:

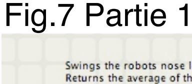
Virage = Kp * (erreur)

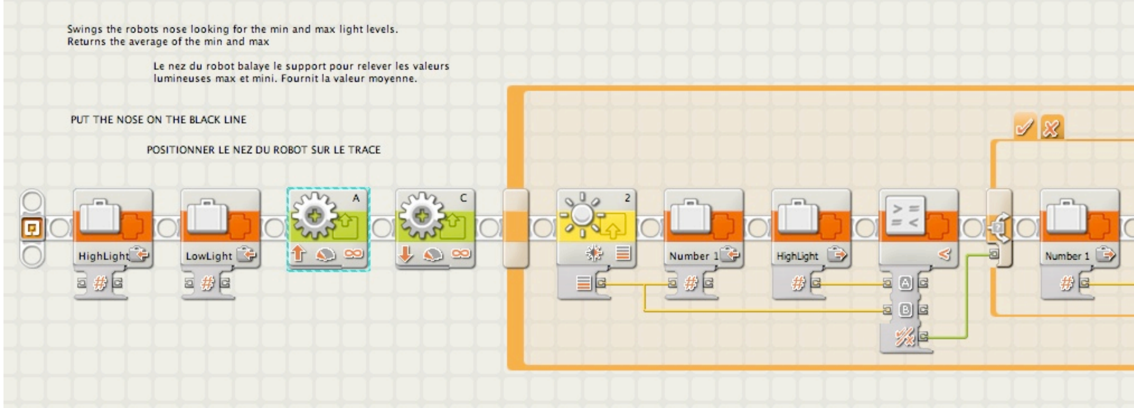
Nous multiplierons **Kp** par 100, ce qui signifie que notre **Virage** calculé est 100 fois plus grand que cela devrait être. Aussi, avant d'utiliser **Virage** nous devons le diviser par 100 pour rectifier.
Ainsi, notre nouveau pseudo-code sera modifié de la manière suivante:

```
Kp = 1000                               ! Souvenez-vous, on emploi Kp*100 aussi il
est réellement égal à 10 !
offset = 45                             ! Initialisation de trois variables
moyenne = 45
Tp = 50
Loop forever                            ! Début de Boucle pour toujours
   LightValue = valeurlueducapteur      ! Quelle est la valeur lue courante du
capteur?
   erreur = LightValue - moyenne        ! calcul de l'erreur par soustraction de
la moyenne.
   Virage = Kp * erreur                 ! le terme "P", de combien veut-on
modifier la puissance du moteur
   Virage = Virage/100                  ! Souvenez-vous,pour annuler l'effet du
facteur 100 dans Kp !
   powerA = Tp + Virage                 ! niveau de puissance pour le moteur A
   powerC = Tp - Virage                 ! niveau de puissance pour le moteur C
   MOTOR A direction=forward power=powerA  ! transmet la commande avec la
nouvelle valeur de puissance dans un bloc MOTEUR.
   MOTOR C direction=forward power=powerC  ! Commande identique à l'autre moteur
mais en utilisant l'autre valeur de puissance.
end loop forever                        ! fin de boucle, retour au début de boucle
et recommencer à nouveau.
```

A - Analyse de la routine 'Mon Bloc' PID_LF_Calib.

Voici à quoi ressemble le programme, et pour une meilleure lecture, ce 'Mon Bloc' sera découpé en 4 parties analysées successivement.

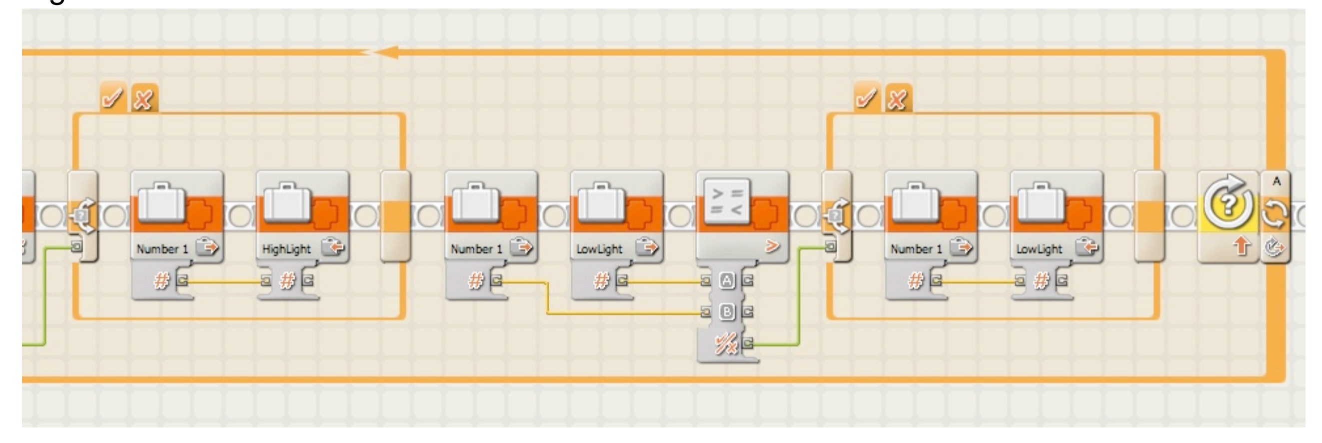

Fig.7 Partie 1



Deux variables numériques sont crées pour stocker les valeurs Hautes (high) et Basses (low) des niveaux lumineux lu par le capteur lors de son balayage. On initialise ces variables en y inscrivant les valeurs suivantes:
-10000 dans High et +10000 dans Low.
Deux Blocs MOTEUR (A et C) avec déplacement opposé pour un balayage lent (fait pivoter le robot) sur lui-même.
Puis on ouvre une boucle dans laquelle on place un Bloc Capteur Photosensible.
Cette boucle est destinée a enregistrer les valeurs lues 'noir' et 'blanc' du test pad en une seule fois.

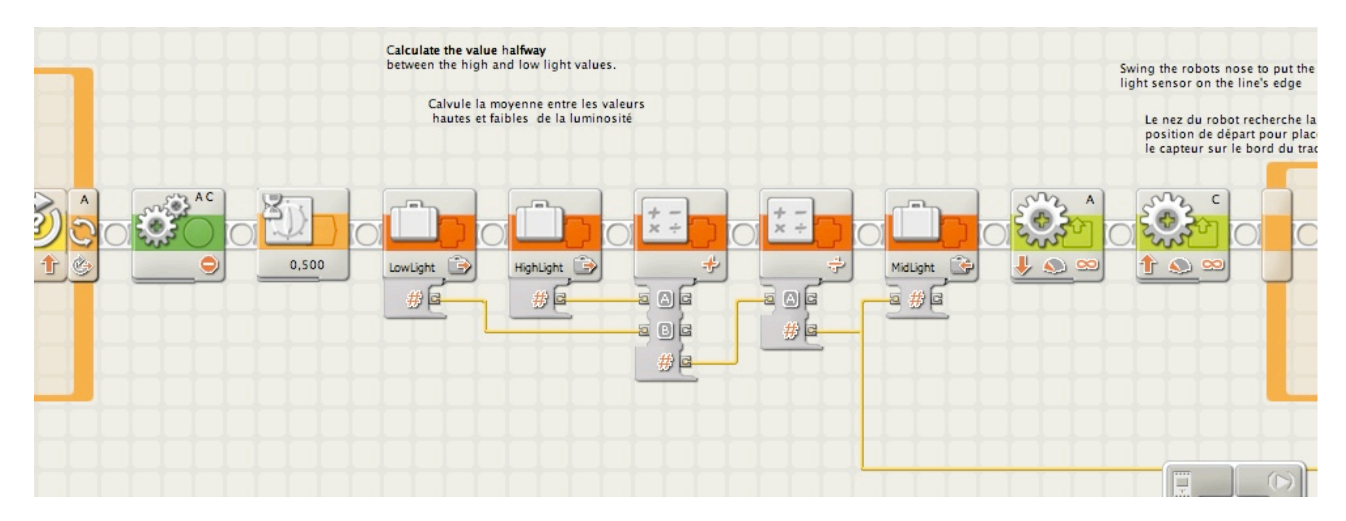
Pour cela on utilise une petite astuce en introduisant une nouvelle variable Number 1 qui recueille directement la valeur lue par le capteur au moment du balayage. Comme le capteur se déplace, Number 1 va recevoir plusieurs valeurs successives annulant à chaque fois la précédente. Il faut donc trouver le moyen de conserver la valeur la plus basse avant qu'elle ne soit remplacée par la valeur haute.
Pour cela, on compare (bloc commutateur) Number 1 d'abord avec HighLigth qui contient -10000. Si HighLigth est inférieur à Number 1 alors on remplace -10000 par Number 1. Puis on passe au 2ème bloc commutateur. en comparant Number 1 avec LowLight qui contient +10000. Si LowLigth est supérieur à Number 1 alors on remplace +10000 par Number 1. Cette boucle est contrôlée par le capteur de rotation du moteur A. Quand le moteur A termine ses 180° de rotation, on quitte la boucle.

Fig.8 Partie 2



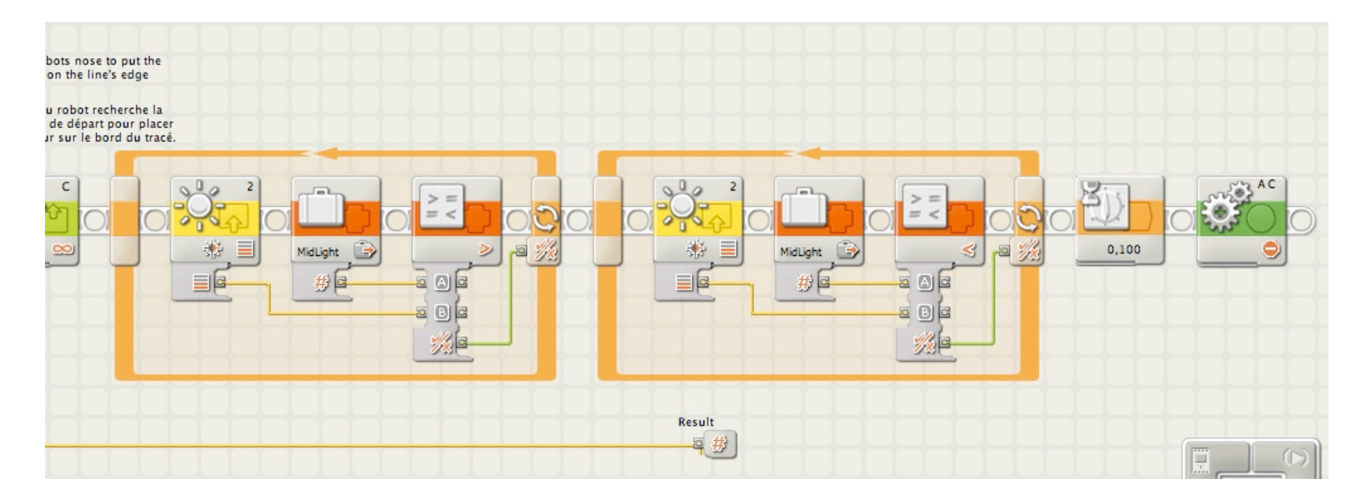
A la sortie de la boucle, le robot s'arrête, puis légère attente avant de procéder au calcul de la demi-somme qui est stockée dans une 4ème variable appelée MidLight. Ce travail réalisé, le robot reprend sa rotation en sens inverse pour revenir vers le tracé.

Fig.9 Partie 3



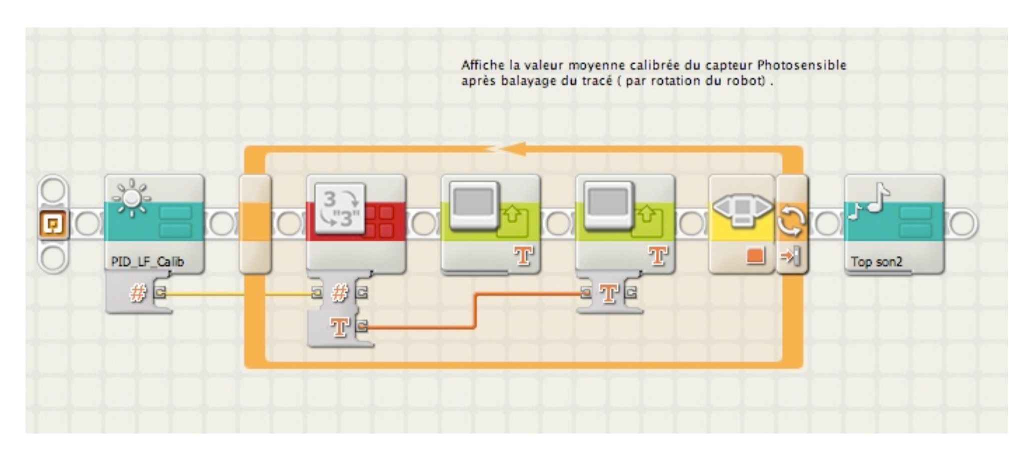
Le capteur lumineux s'assure que le niveau est supérieur à la moyenne (MidLight) c'est à dire qu'il est toujours sur le 'blanc' (1ère boucle) et le robot poursuit sa rotation en passant sur le 'noir' (2éme boucle). Si le niveau est inférieur à MidLight, on est alors sûr d'être sur le 'noir'. Dès que le capteur ne détecte plus le 'noir', on sort de la boucle, petite attente et le robot s'arrête.

Fig.10 Partie 4



Remarques:
1 - La fin de course se produit sur le 'blanc', après avoir franchi le bord du tracé.
2 - Vous observerez un plot de fil de données un peu spécial (il est "flottant") provenant d'une sortie de la variable MidLight; il porte le nom de Result et contient le signe # qui est celui d'une valeur numérique. Ce type de plot apparaît lorsque le programme devient un 'Monbloc' caractérisé par une bande verte. Donc, si vous souhaitez compléter ce programme par un affichage, il suffira de créer un nouveau programme conçu par exemple de cette manière:

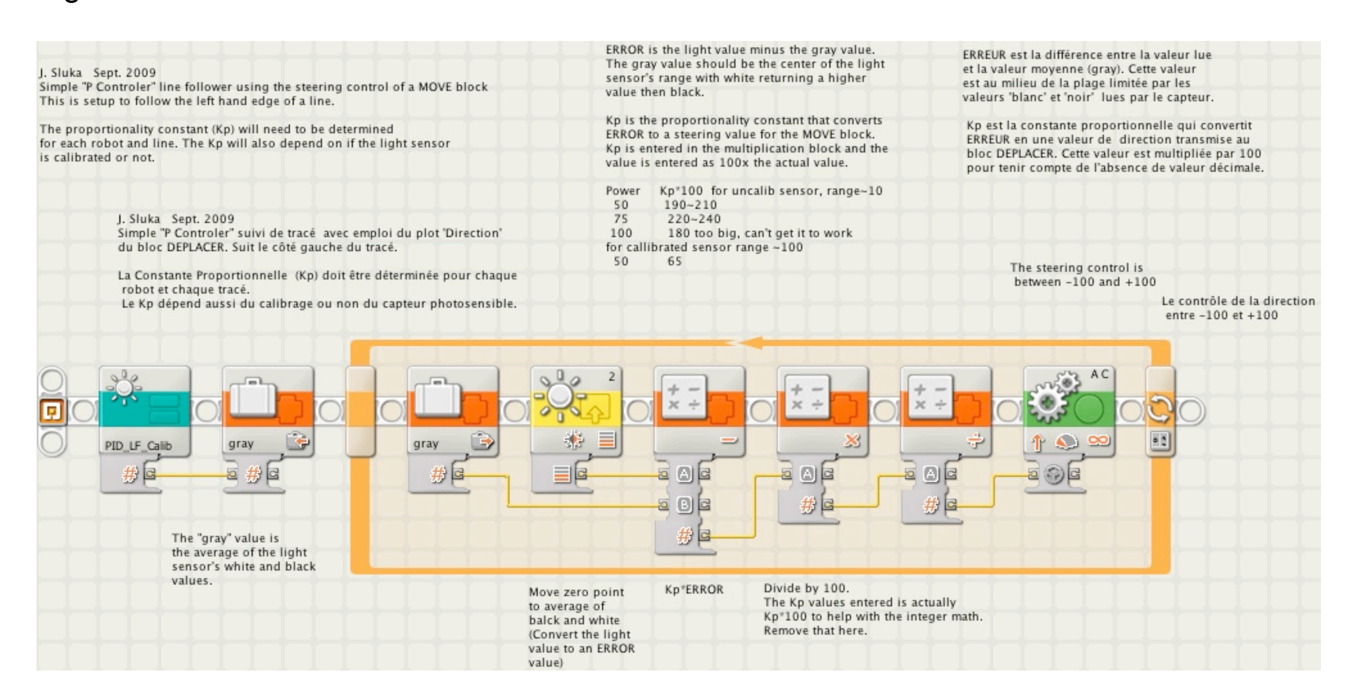
Fig.11



B - Analyse du programme principal.

Nous disposons maintenant de tous les éléments nécessaire pour l'écriture du **contrôleur P**. Le programme est intitulé *Steer_LF_2.rbt.*

Fig.12



Comme vous pouvez le constater, grâce au 'Monbloc' PID_LF_Calib, le programme par lui-même est assez simple.
La variable 'Gray' reprend la valeur moyenne calculée à partir de ce 'Mon Bloc' . La boucle se contente de relever la valeur du capteur pendant le déplacement du robot; de cette valeur est déduite la valeur 'Gray' pour obtenir la valeur de l'Erreur. Ce résultat est ensuite multiplié par la constante Kp (multiplié par le facteur 100 pour résoudre le problème des valeurs décimales)pour obtenir la ***correction de puissance*** à appliquer aux moteurs. Le résultat est ensuite divisé par 100 pour retrouver les valeurs de la plage acceptée par les moteurs. Le fil de données est en final raccordé au *plot 'Direction' du bloc DEPLACER.* A ce sujet, vous noterez que la plage s'étend de - 100 à + 100. Quand le fil de données produit une valeur égale à zéro, le robot se déplace en ligne droite. Si la valeur est entre - 100 et zéro, le robot vire à gauche; il vire à droite si la valeur est entre zéro et + 100. Les virages sont plus ou moins serrés selon la valeur absolue transmise.

Remarque: lors du premier essai, vous serez probablement déçu du résultat. Cela signifie que des corrections sont nécessaires, puisque Kp et la puissance des moteurs sont variables.
Comme il a été signalé plus haut, vous procéderez par correction de ces deux paramètres jusqu'à trouver les valeurs qui produisent le meilleur résultat.

N'oubliez pas de créer et d'installer dans 'Mon Bloc' le programme *PID_LF_Calib.rbt* ***avant de créer*** le programme *Steer_LF_2.rbt.*

Vous pouvez télécharger ces programmes ici:
PID_LF_Calib.rbt
files.me.com/roboleo/t4kr7s

Steer_LF_2.rbt
files.me.com/roboleo/0vgi8s

CHAPITRE 3

Ajouter "I" au contrôleur : le contrôleur PI

Pour améliorer les performances de notre contrôleur **P** nous ajouterons un nouveau terme à l'équation. Ce terme est appelé **l'intégrale**, le "**I**" dans **PID**. Les Intégrales sont une partie très importante des mathématiques supérieures, heureusement pour nous, celle dont nous avons besoin est assez simple.

L'intégrale est la somme courante des erreurs.

En effet, c'est aussi simple. Il y a toutefois quelques subtilités que nous laisserons sous silence pour l'instant.

Chaque fois que nous lirons une valeur fournie par le capteur lumineux et calculerons **l'erreur** qui en découle, nous ajouterons cette erreur à une variable nous appellerons intégrale (intelligent non ?).

Intégrale = intégrale + erreur

Cette équation paraît bizarre et elle l' est. Elle n'est pas écrite comme une équation mathématique courante, mais sous une forme communément utilisée en programmation pour additionner une série de valeurs. Mathématiquement cela n'a aucun sens.
Dans la programmation informatique le signe égal (=) a une signification quelque peu différente que celle des maths. C'est une expression de programmation et non pas une formule mathématique appropriée.
Le signe " = " signifie en mathématiques que l'expression située à droite du signe permet de calculer et de ranger le résultat dans une variable située à gauche.
Ce que nous demandons à l'ordinateur, c'est d'ajouter à l'ancienne valeur d'intégrale , celle de l'erreur et de ranger le résultat à la place de l'ancienne intégrale.

Ensuite, comme pour le terme proportionnel **P**, nous multiplierons **l'intégrale** par une constante, c'est un autre terme **K**. Puisque cette constante est associée avec le terme *intégral* nous l'appellerons **Ki**. Comme pour l'expression proportionnelle précédente, nous multiplions *l'intégrale* par la constante **(Ki)** pour obtenir une correction.

Ce qui donne à notre précédente formule du *Virage* un complément additionnel, et la modifie ainsi:

Virage = Kp * (erreur) + Ki * (Intégrale)

Nous retrouvons ici l'équation de base pour un contrôleur **PI**. Le **Virage** est la correction pour les moteurs. Le terme proportionnel est **Kp** * (l'erreur) et le terme intégrale est **Ki** * (intégrale).

A quoi sert exactement le terme intégrale? Si **l'erreur** garde le même signe pendant plusieurs boucles **l'intégrale** croît de plus en plus.
Par exemple, si nous vérifions le capteur lumineux et calculons que l'erreur est 1, et qu'un peu plus tard nous vérifions à nouveau le capteur et l'erreur est 2, et la fois suivante encore 2, alors l'intégrale sera 1 + 2 + 2 = 5. *L'intégrale* est égale à 5 alors que *l'erreur* à cette étape particulière est seulement de 2. On voit que *l'intégrale* peut être un grand facteur dans la correction mais il lui faut un certain temps avant qu'elle ne puisse commencer à contribuer.
Mais l'intégrale fait aussi autre chose; elle supprime les petites erreurs. Si dans notre suivi de tracé le capteur lumineux est assez proche du bord du tracé, mais pas exactement dessus, l'erreur sera sera petite et la correction de trajectoire sera faible. On serait alors tenté de modifier le **Kp** proportionnel mais cela mènera souvent le robot à serpenter (des oscillations dans les deux sens). Le terme intégrale est parfait pour rectifier les petites erreurs.
Alors que **l'intégrale** additionne les erreurs, plusieurs petites erreurs consécutives peuvent finalement lui donner assez d'importance pour faire une différence.
En d'autres termes, l'intégrale est une sorte de "mémoire" du contrôleur. Elle est l'histoire cumulative des erreurs et donne au contrôleur un moyen de les corriger quand elles persistent pendant une longue période de temps.

Quelques subtilités sur les intégrales

Bien, mais l'intégrale est un peu plus compliquée. Heureusement ce n'est pas trop douloureux.
J'ai ignoré un paramètre secondaire (ce n'est pas vraiment secondaire, mais nous allons le considérer ainsi), le temps. L'intégrale est vraiment la somme de **l'erreur** * *(l'écart de temps)*. L'écart de temps ou delta (**dT**) est le temps écoulé entre la dernière lecture du capteur et le temps du contrôle le plus récent du capteur;

Intégrale = intégrale + erreur * (dT)

Ainsi, à chaque fois que nous ajoutons à **intégrale** un élément, nous devrions lui ajouter les temps **d'erreur** le **dT**.
Il est assez facile de faire mesurer le **dT** par le robot. Il suffit d'utiliser un *minuteur* à chaque relevé du capteur photosensible. Si nous soustrayons la dernière lecture du temps actualisé nous obtenons le temps écoulé depuis la dernière lecture: le **dT**. (Il existe une meilleure façon de le faire mais je l'ignorerai puisque cela n'est pas nécessaire). Mais ne serait-il pas plus simple s'il n'y avait pas la nécessité de mesurer le **dT** pour faire la multiplication ?
Bien, et si le **dT** avait toujours la même valeur? Chaque fois que **intégrale** croît, nous avons ce même terme **dT**. Donc nous pouvons faire abstraction de ce facteur **dT** de **l'erreur * (dT**) et se contenter d'appliquer une formule analogue du type:

intégrale = intégrale + erreur

Seule restriction, quand nous ferons un autre calcul avec **intégrale** nous devrons en réalité multiplier le résultat par **dT**.
Mais attendez il y a encore plus...

Nous pouvons faire mieux pour faire abstraction de l'expression du temps. Le terme **intégrale** dans l'équation du contrôleur de **PI** est **Ki * (intégral)** * **dT**. Mais **Ki** est un paramètre qui doit être réglé avec précision (tout comme **Kp**), alors pourquoi ne pas remplacer juste la partie **Ki*dT** par un nouveau **Ki** ? Ce nouveau **Ki** diffère du premier, mais puisque nous ne le connaissons pas, qu'importe celui que nous utilisons ou la manière de le désigner. Peu importe sa nomination ou ce qu'il représente nous devons toujours trouver la valeur correcte en grande partie par tâtonnement.

Nous avons donc supprimé l'élément temps pour le terme **intégrale** avec la seule restriction qu'à toutes les étapes, les dT sont constants (gardent la même valeur).

L'intégrale a une mémoire d'éléphant

Un dernier détail devrait être mentionné au sujet de **l'intégrale**. D'ordinaire, **l'intégrale** tend vers zéro, c'est-à-dire qu'elle ne contribue en rien au contrôleur, puisque des valeurs **d'erreur** cumulées qui sont de signes opposés pour la plupart d'entre elles, sont déjà rassemblés dans **intégrale**.
Par exemple, si après plusieurs cycles les **erreurs** respectives sont égales à 1,2,2,3,2,1 cela donne une **intégrale** de 11. Mais **l'erreur** au dernier cycle de données n'est seulement que 1, valeur qui est beaucoup plus petite que **l'intégrale** en ce point. La seule façon pour **l'intégrale** de tendre vers zéro, est d'obtenir une suite **d'erreurs** négatives pour contre balancer la suite précédente **d'erreurs** positives et atteindre cette limite. Par exemple, si les prochaines '**erreurs** sont-2,-2,-3 alors l'intégrale baissera de 11 à 4 et nous aurions toujours besoin de plus d'erreurs négatives pour atteindre zéro . De plus, **l'intégrale** demande que l'erreur totale soit également répartie entre des erreurs positives et négatives.

Si un événement dévie notre robot vers la gauche du bord du tracé, le terme **intégral** cherche non seulement à le faire revenir sur ce bord, mais aussi le faire dépasser vers la droite de la même valeur que celle de la gauche. Ainsi **l'intégrale** a tendance à atteindre par le haut cette limite, si de grandes **erreurs** persistent quelque temps. Cela peut aussi causer quelques problèmes avec les contrôleurs qui incluent un terme **intégral**. Parfois cette tendance de dépassement du terme **intégral** dans sa tentative de correction **d'erreur** oblige le programmeur à intervenir pour neutraliser ces problèmes. Si la limite supérieure de **l'intégrale** constitue un problème deux solutions sont envisageables:
(1) remettre à zéro la variable **intégrale**, chaque fois que **l'erreur** est égale zéro ou à chaque changement du signe **d'erreur**.
(2) "Réduire" **l'intégrale** en multipliant sa valeur cumulée par un facteur inférieur à 1 quand un nouvelle valeur intégrale est calculée. Par exemple:

intégrale = (2/3) * intégrale + erreur

Cela réduit la valeur **intégrale** précédente de 1/3 à chaque boucle. Si vous considérez le terme **intégrale** comme la mémoire du contrôleur, alors cet diminution le force à "oublier" des événements qui se sont produits dans un passé lointain.

Pseudo code pour le contrôleur PI

Pour ajouter le terme intégral au contrôleur, nous avons besoin d'ajouter une nouvelle variable pour **Ki** et une pour **l'intégrale** proprement dite. Et ne pas oublier que nous multiplions Ks par 100 pour résoudre le problème des valeurs décimales.

```
Kp = 1000                            ! Souvenez-vous, on emploi Kp*100 aussi il
est réellement égal à 10 !
Ki = 100                             ! Souvenez-vous, on emploi Ki*100 aussi il
est réellement égal à 1 !
offset = 45                          ! Initialisation de trois variables
moyenne = 45
Tp = 50
integral = 0                         ! l'endroit où nous stockerons notre
intégrale
Loop forever                         ! Début de Boucle pour toujours
   LightValue = valeurlueducapteur   ! Quelle est la valeur lue courante du
capteur?
   erreur = LightValue - moyenne     ! calcul de l'erreur par soustraction de
la moyenne.
 integral = integral + error         ! notre nouveau terme intégral
Virage = Kp * erreur + Ki * integral  ! le terme "P", et le terme "I"
   Virage = Virage/100               ! Souvenez-vous,pour annuler l'effet du
facteur 100 dans Kp !
   powerA = Tp + Virage              ! niveau de puissance pour le moteur A
   powerC = Tp - Virage              ! niveau de puissance pour le moteur C
   MOTOR A direction=forward power=powerA ! transmet la commande avec la
nouvelle valeur de puissance dans un bloc MOTEUR.
   MOTOR C direction=forward power=powerC ! Commande identique à l'autre moteur
mais en utilisant l'autre valeur de puissance.
end loop forever                     ! fin de boucle, retour au début de boucle
et recommencer à nouveau.
```

Ajouter " D " au contrôleur : le contrôleur PID au complet

(" D ": que va-t-il se produire maintenant?)

Notre contrôleur dispose maintenant d'un terme **proportionnel** (P) qui cherche à corriger **l'erreur** *courante* et un terme **intégral** (I) qui cherche à corriger les **erreurs** *passées*. A présent, existe-t-il un moyen qui permette au contrôleur de prévoir à l'avance et peut-être d'essayer de corriger les **erreurs** qui ne se sont pas encore produites?

La réponse est oui, et la solution est un autre concept des mathématiques supérieures appelé la **dérivée**. Ah, nous y voilà; c'est le fameux **D** du **PID.** Tout comme **l'intégrale**, la **dérivée** représente aussi une autre partie très importante des mathématiques supérieures, heureusement pour nous, ce dont nous avons besoin pour le **PID** est assez simple.
Nous pouvons prévoir le futur en supposant que la prochaine variation de **l'erreur** est identique à la dernière variation de **l'erreur**.

Cela signifie que la prochaine **erreur** est supposée être **l'erreur** *courante* plus (+) la *variation* de **l'erreur** entre deux précédentes lectures du capteur. La variation de l'erreur entre deux points consécutifs est appelée **dérivée**. La **dérivée** est équivalente à la pente d'une droite.

Cela pourrait paraître un calcul un peu complexe, mais ce n'est vraiment pas le cas ici. Un jeu de données type aidera à mieux comprendre le mécanisme. Supposons que **l'erreur** *actuelle* est 2 et **l'erreur** *précédente* 5. Quelle pourrait-être **l'erreur** *suivante*? Bien, la *variation* de **l'erreur** est la **dérivée** qui s'exprime ainsi:

(**l'erreur** *actuelle*) - (**l'erreur** *précédente*)
ce qui donne pour nos valeurs: 2 - 5 = - 3. La **dérivée** actuelle est dans ce cas - 3. Pour utiliser la **dérivée** afin de prévoir **l'erreur** *future*, nous utiliserons la relation suivante:

(**l'erreur** *future*) = (**l'erreur** *actuelle*) + (**dérivée** actuelle)

qui dans notre cas donnera: 2 + (- 3) = - 1. Nous supposerons donc que l'erreur future sera égale à - 1.
En pratique nous n'allons pas systématiquement calculer **l'erreur** *suivante*. Au lieu de cela nous utiliserons la **dérivée** directement dans l'équation de contrôleur.

Le terme **D**, tout comme le terme **I** , devrait en réalité inclure un élément temps, et le terme 'officiel' de **D** serait:

Kd(dérivée) / (dT)

Tout comme les termes **proportionnel** et **intégrale** nous devons le multiplier par une constante. Puisque cette constante accompagne la **dérivée,** nous l'appellerons **Kd**. Vous remarquez que pour le terme **dérivée** nous le *divisons* par **dT**, tandis que le terme **intégrale** était multiplié par **dT**. Ne vous inquiétez pas trop de cela puisque nous allons adopter la même astuce pour nous débarrasser du **dT** du terme **dérivé,** tout comme nous avons fait pour le terme **intégrale**. La fraction **Kd/dT** est une constante si **dT** est le même pour chaque boucle. Donc nous pouvons remplacer **Kd/dT** par un autre **Kd**. Alors que ce **K**, comme le **Ks** précédent, est inconnu et doit être déterminé par tâtonnement, cela n'a pas d'importance si l'on adopte **Kd** au lieu de **Kd/dT**.

Nous pouvons maintenant écrire l'équation complète du contrôleur PID:

Virage = Kp * (erreur) + Ki * (Intégrale) + Kd * (dérivée)

Il est assez évident que "la prévision de l'avenir" serait une chose utile à réaliser, mais comment s'y prendre? Et comment déterminer la précision de cette prévision ?

Si **l'erreur** actuelle est plus grande que **l'erreur** précédente alors le terme **D** essayera de corriger cette **erreur**. S'il **l'erreur** actuelle est plus petite que **l'erreur** précédente alors le terme **D** essayera de stopper le contrôleur dans sa tentative de correction **d'erreur**.
C'est ce deuxième cas qui est particulièrement intéressant. Si **l'erreur** s'approche du zéro, alors nous tendons vers le point d'arrêt de la correction. Comme le système prend quelque temps pour répondre aux changements de la puissance des moteurs, nous voulons commencer à réduire la puissance du moteur avant que **l'erreur** ne tende en réalité vers zéro, sans quoi nous risquons le dépassement. Dans ces conditions, il semblerait que l'équation pour le terme **D** soit plus complexe, mais en vérité, elle ne l'est pas. La seule chose dont vous devez vous soucier est de procéder à la soustraction dans le bon ordre. L'ordre correct pour ce type d'opération est "actuel" moins "précédent". Ainsi pour calculer la **dérivée** nous prenons **l'erreur** actuelle et soustrayons **l'erreur** précédente.

Pseudo codez pour le contrôleur PID

Pour ajouter le terme dérivée au contrôleur, nous avons besoin d'ajouter une nouvelle variable pour **Kd** et une variable pour conserver la dernière **erreur**. Et ne pas oublier que nous multiplions Ks par 100 pour résoudre le problème des valeurs décimales.

```
Kp = 1000                           ! Souvenez-vous, on emploi Kp*100 aussi il
est réellement égal à 10 !
Ki = 100                            ! Souvenez-vous, on emploi Ki*100 aussi il
est réellement égal à 1 !
Kd = 10000                          ! souvenez-vous, on emploi Kd*100 aussi il
est réellement égal à 100
offset = 45                         ! Initialisation de trois variables
moyenne = 45
Tp = 50
integral = 0                        ! l'endroit où nous stockerons notre
intégrale
lastError = 0                       !l'endroit où nous stockerons la valeur de
l'erreur précédente
derivative = 0                      ! l'endroit où nous stockerons notre
dérivée

Loop forever                        ! Début de Boucle pour toujours
   LightValue = valeurlueducapteur  ! Quelle est la valeur lue courante du
capteur?
   erreur = LightValue - moyenne    ! calcul de l'erreur par soustraction de
la moyenne.
  integral = integral + error       ! notre nouveau terme intégral
derivative = error - LastError      ! calcule la valeur dérivée

Virage = Kp * erreur + Ki * integral + Kd*derivative        ! le terme "P", et
le terme "I", ainsi que le terme "D"
   Virage = Virage/100              ! Souvenez-vous,pour annuler l'effet du
facteur 100 dans Kp !
   powerA = Tp + Virage             ! niveau de puissance pour le moteur A
   powerC = Tp - Virage             ! niveau de puissance pour le moteur C
   MOTOR A direction=forward power=powerA  ! transmet la commande avec la
nouvelle valeur de puissance dans un bloc MOTEUR.
   MOTOR C direction=forward power=powerC  ! Commande identique à l'autre moteur
mais en utilisant l'autre valeur de puissance.
lastError = Error                   ! enregistre la nouvelle valeur de
l'erreur qui sera la lastError au prochain cycle.
end loop forever                    ! fin de boucle, retour au début de boucle
et recommencer à nouveau.
```

Nous avons maintenant établi pour notre contrôleur **PID**, le pseudo-code complet du robot suiveur de tracé .
Reste à présent à aborder la partie la plus délicate du **PID**: le réglage. Il s'agit en fait de déterminer les valeurs optimales à attribuer aux constantes **Kp, Ki et Kd.**

CHAPITRE 4: réglage d'un contrôleur PID

Sans faire usage aux calculs complexes (mais nous devons tout de même faire un peu de mathématiques).

Nous abordons ici un aspect de la question qui nous réserve des surprises. Ou plutôt nous choisirons la méthode expérimentale, qui est celle des chercheurs. C'est par répétition et

essais multiples, par approximations successives que nous procéderons aux réglages. Les constantes sont liées à trop de paramètres variables pour être fixées une fois pour toutes.

En clair, il faudra faire des séries d'essais en modifiant les paramètres.

Les personnes averties ont déjà compris comment régler un contrôleur PID.
Pour ma part, n'étant pas aussi doué, j'utiliserai ce qu'ils ont appris. Il s'avère que le relevé d'un couple de paramètres permet de calculer des valeurs pour **Kp, Ki et Kd** avec une précision suffisante.
Peu importe le système qui contrôle les équations de réglage, elles fonctionnent presque toujours correctement. Il y a plusieurs techniques pour calculer le **Ks**, l'une d'entre elles est appelée la "Méthode Ziegler-Nichols"; c'est celle que nous utiliserons. Une recherche sur Google offrira de nombreuses pages Web qui décrivent cette technique. La version utilisée est extraite de la page de 'Wiki page on PID Controllers' (le même traitement est trouvé en beaucoup d'autres endroits). Il y aura juste une petite modification en insérant la boucle temps (**dT**) dans les calculs figurants dans la table ci-dessous.

Pour régler votre Contrôleur PID,

suivez ces étapes:

1 - Fixez la valeur zéro aux constantes **Ki** et **Kd**. Cela force le contrôleur à se conduire comme un simple contrôleur **P**.
2 - Attribuez au terme **Tp** une valeur la plus faible possible. Pour nos moteurs, 25 serait une bonne valeur de début.
3 - Attribuez au terme **Kp** une valeur "raisonnable". Qu'est-ce qui est "raisonnable" ?
a) choisissez la valeur maximale du contrôle de puissance du moteur (100) et divisez la par la valeur d'erreur utilisable maximale. Pour notre robot suiveur de tracé nous avons supposé que l'erreur maximale était de 5, donc notre projection pour **Kp** est 100/5=20. Quand l'erreur est égale à + 5 la puissance du moteur sera de 100 unités. Quand l'erreur sera égale à zéro la puissance du moteur sera assise au niveau de **Tp**.
b) Ou alors, fixez la valeur de **Kp** à 1 (ou 100) et notez le résultat.
c) Si vous avez introduit un multiplicateur pour les **K's** de 100 fois leur valeur réelle, vous devez en tenir compte ici. 1 sera saisi comme 100, 20 comme 2000, 100 comme 10000.

4 - Lancez le programme et observez le comportement du robot. S'il ne peut pas suivre le tracé et se déplace à l'aveuglette, alors augmentez la valeur de **Kp**. S'il oscille d'une manière extravagante alors diminuez la valeur de **Kp**. Continuez à modifier la valeur de **Kp** jusqu'à ce que vous trouviez celle qui fournit une oscillation sur le tracé acceptable. Nous appellerons ce **Kp** "**Kc**" ("le gain critique" dans la littérature PID).
5 - Utilisez la valeur **Kc** comme **Kp**, Lancez le programme du robot et essayez de déterminer la vitesse de l'oscillation. Cela peut être délicat, mais heureusement le relevé n'a pas besoin d'une grande précision. La période d'oscillation (**Pc**) représente le temps nécessaire au robot pour balayer de part et d'autre du tracé et revenir à son point de départ. Pour des robots Lego **Pc** sera probablement dans la plage d'environ 0.5 seconde à une seconde ou deux.
6 - Vous devez aussi connaître à quelle vitesse tournent les cycles du robot au travers de la boucle de contrôle. On a ici donné arbitrairement à la boucle un nombre fixe de tours (comme 10,000) ainsi que le temps nécessaire au robot pour accomplir ce travail (ou demander au robot de chronométrer et afficher le résultat.) le temps par boucle (**dT**) est le

temps mesuré divisé par le nombre de boucles. Pour un contrôleur PID complet, programmé en NXT-G, sans aucun bloc de sons, le **dT** sera dans la plage de 0.015 à 0.020 secondes par boucle.
7 - Utilisez la table ci-dessous pour calculer une série de valeurs **Kp**, **Ki** et **Kc**. Si vous voulez juste un contrôleur **P** utilisez alors la ligne dans la table marquée **P** pour calculer la valeur "correcte" **Kp** (**Ki** ' et **Kd** ' ont tous les deux une valeur zéro). Si vous ne voulez qu'un contrôleur **PI** utilisez alors la ligne suivante. Le contrôleur **PID** complet est le résultat final(la ligne inférieure).

8 - Si vous avez déjà multiplié K par 100 pour tenir compte de l'absence de valeurs décimales, ignorez cette opération pour les calculs. Ce facteur de 100 est déjà pris en compte dans **Kp** = **Kc** valeur que vous avez déterminé.
9 - Lancez le programme et observez le comportement du robot.
10 - Faîtes varier la valeur **Kp**, **Ki** et des valeurs de **Kd** pour obtenir la meilleure performance possible. Vous pouvez commencer par une assez grande variation, disons 30 % puis essayez alors une plus petite afin d'obtenir une valeur optimale (ou au moins acceptable) de la performance.
11 - Une fois réunis un bon jeu d'essais de **K's** d'augmenter la valeur de **Tp**, qui contrôle la vitesse rectiligne du robot.
12 - Réajustez **K's** ou peut-être revenez à l'étape 1 et répéter le processus en entier pour une nouvelle valeur de **Tp**.
13 - Répéter a nouveau jusqu'à ce que le comportement du robot vous convienne.

Méthode Ziegler-Nichols

Fig 7

Méthode Ziegler–Nichols pour les valeurs *K*' (les boucles temps sont considérées constantes et égales a dT)			
Control Type	**Kp**	**Ki'**	**Kd'**
P	0.50*Kc*	0	0
PI	0.45*Kc*	1.2*Kp*dT/ *Pc*	0
PID	0.60*Kc*	2*Kp*dT / *Pc*	*KpPc* / (8dT)

La virgule (en exposant) sur les **Ki's** et **Kd's** sont là juste pour rappeler qu'ils sont calculés en supposant **dT** constant et compris dans les valeurs de **K**.

James Sluka, auteur de cet article, a testé pour son robot certaines valeurs avec les commentaires suivants:
Pour **Kc** = 200 et quand **Kp** = **Kc**, la fréquence des oscillations du robot était de 0,8 seconde. Donc **Pc** = 0,8.
Il a mesuré **Pc** en comptant à haute voix chaque fois que le robot oscillait totalement dans une direction particulière. Il a alors comparé sa perception du comptage à " 1 potato, 2 potato, 3 potato…". Ce n'est peut-être pas de la mécanique de précision, mais cela fonctionne assez bien dans ce que nous appellerons "l'ingénierie pratique". La boucle de temps **dT**, est de 0.014 secondes par cycle déterminée en exécutant simplement le programme 10,000 fois et en affichant sur l'écran du NXT le temps écoulé. En utilisant la table ci-dessus pour un contrôleur PID on arrive au résultat suivant:

Kp = (0.60)(Kc) = (0.60)(200) = 120
Ki = 2(**Kp**)(**dT**) / (**Pc**) = 2(120)(0.014) / (0.8) = 4.2 (arrondi à 4)
Kd = (**Kp**)(**Pc**) / ((8)(**dT**)) = (120)(0.8) / ((8)(0.014)) = 857

Après plusieurs essais et réglages, les valeurs suivantes ont été retenues: 220, 7, et 500 pour respectivement **Kp, Ki, et Kd**. Sans oublier que tous les **K's** étaient affectés du facteur 100, ce qui signifie que les valeurs réelles sont respectivement 2,2 - 0,07 - et 5.

Comment les variations des Kp, Ki, et Kd modifient les performances des robots.

Le tableau et la méthode décrits ci-dessus constituent un bon point de départ pour optimiser votre **PID**. C'est aussi une aide pour une meilleure compréhension du résultat selon l'augmentation ou la diminution de l'un des trois **Ks**. La table ci-dessous est disponible sur beaucoup de sites Web. La version particulière ci-après est celle du Wiki sur des contrôleurs **PID**.

Fig. 8

Effets sur l'accroissement des paramètres				
Paramètres	**Temps de réactivité**	**Dépasseme nt**	**variation du temps**	**Erreur à l'équilib.**
K_p	Décroissant	Croissant	Petit changemt	Décrois.
K_i	Décroissant	Croissant	Croissant	Supprimé
K_d	Non défini (petite crois. ou décrois.)	Décrois.	Décrois.	Néant

Le "temps de réactivité" correspond à la vitesse de rectification de l'erreur par le robot. Dans notre cas type il correspond à la vitesse du robot dans sa tentative de revenir au bord du tracé après s'en être éloigné. Le temps de réactivité est surtout contrôlé par **Kp**. Plus **Kp** est grand, et plus le robot accélére ce retour en diminuant le temps de réactivité. Si **Kp** est trop grand le robot perd le contrôle..

"Le Dépassement" est à la distance d'écartement (par rapport au bord du tracé) que le robot a tendance à prendre alors qu'il réagit à une erreur. Par exemple, si le dépassant est petit, le robot ne vire pas à droite du tracé, il essaye alors de rectifier en virant à gauche du tracé. Si le dépassement est grand, alors le robot oscille devant le bord du tracé et essaye de corriger l'erreur. Le dépassement est en grande partie contrôlé par le terme **Kd**, mais est fortement affecté par les Termes **Kp** et **Ki.** D'habitude pour corriger fortement le dépassement vous serez tenté d'augmenter **Kd.**
Rappelez-vous notre premier suivi de tracé le plus simple, celui qui ne fait rien d'autre que virer à droite ou gauche ? Ce dispositif a un très mauvais dépassement.

"Variation du temps" est le temps pris par le robot pour prendre en compte un grand changement. Dans notre cas, un grand changement survient quand le robot procède à un virage**.** Alors que le robot répond à la courbe, il corrigera **l'erreur** puis le dépassement par une certaine valeur. Il doit alors corriger ce dépassement et revenir dans l'autre sens.
Il doit alors corriger ce dépassement… bien, vous avez l'idée. Pour réagir à une erreur le robot aura tendance à osciller autour de la position souhaitée. **"Variation du temps"** est le temps pris par cette oscillation pour revenir à zéro. "Variation du temps" répond fortement tant au terme de **Kd** qu'à celui de **Ki**. Plus **Ki** est grand et plus "Variation du temps" est long. Plus **Kd** est grand et plus "Variation du temps" est court.

"L'Erreur à l'Équilibre" est l'erreur restante alors que le dispositif fonctionne sans être dérangé. Pour notre suivi de tracé, ce serait comme suivre une trajectoire rectiligne. Les contrôleurs **P** et **PD** résolvent souvent cette sorte d'erreur. Elle peut être réduite en augmentant **Kp** mais cela peut déclencher une oscillation du robot. Incluant un terme **I** et en augmentant **Ki** on corrigera souvent un contrôleur P ou PD qui conserve une erreur constante à l'équilibre. (Cela suppose que vous vous souciez même d'une petite erreur restante alors que le robot suit le tracé. Cela se traduit par une petite correction d'un côté ou d'un autre par une petite valeur).

Chapitre 5: un programme complet d’un contrôleur PID

Pour un robot suiveur de tracé muni d'un simple capteur lumineux.

Nous abordons enfin l'élaboration du programme en NXT-G.

Rappelons rapidement les caractéristiques du dispositif.
Ce contrôleur est un suiveur de ligne gauche" (*left hand line follower*) et dispose d'un moteur A à gauche et un moteur C à droite. Le capteur lumineux est connecté au port 2.

Nous utiliserons 4 programmes NXT-G pour résoudre le problème:
3 MonBlocs qui devront être rangés dans la palette "perso" (songez à les installer).
Un Programme principal qui contiendra ces 3 Monblocs.

1) Le MonBloc **PID_LF_Calib** mesure le niveau lumineux maximum and minimum et calcule le niveau de "gris" (déjà vu au chapitre 2).
Ce Bloc peut-être supprimé et la valeur "gray" écrite directement dans la variable "TargLight".

2) Le MonBloc **LF_PID** effectue les calculs du PID. En entrée on trouve l'erreur courante, et en sortie la correction du niveau de puissance des moteurs (intitulé "PID"). Les Ks sont envoyés au bloc LF_PID comme des variables globales (variables définies à la fois dans le programme principal et dans le MonBloc utilisant le même nom et le même type).

3) Le MonBloc qui contrôle les moteurs (**PID_LF_MotorControl**) utilise les variables "A-motor" et "B-motor" pour respectivement les moteurs gauche et droit. Les moteurs présents contrôlés sont A et C (pauvre choix pour des noms de variables. Il aurait mieux valu les appeler Left_motor et Right_motor).

Le MonBloc **PID_LF_MotorControl** prend en entrée la valeur du PID (de LF_PID) au port "Turn" et la puissance moteur cible (au port "Target"), calcule le niveau de puissance actuel pour chaque moteur et adresse ces valeurs aux moteurs. Si le calcul du niveau de puissance est de signe négatif, il est inversé et envoyé à un Bloc MOTOR pour une rotation inversée (le Bloc MOTOR ne comprend pas une puissance négative signifiant une rotation inverse).
Les 3 constantes (Kp, Ki et Kd) sont inscrites dans les variables valises. De plus, les niveaux "TargLight" et "TargPower" sont également inscrits dans leurs valises respectives.
La valeur "TargLight" est la valeur moyenne (entre le blanc et le noir lus par le capteur).
Le "TargPower" est le niveau de puissance moyenne pour les 2 moteurs lorsqu'ils se déplacent en ligne droite.

Enfin,

PID_lineFol_4 est le programme principal. La boucle principale est calibrée pour un nombre de cycles déterminé. A la sortie de cette boucle, le temps en millisecondes est affiché sur l'écran du NXT ainsi que le
le nombre de rotations du moteur A (en degrés). Le temps écoulé et le nombre de boucles sont utilisés pour calculer la durée d'un cycle (dT), très utile pour calibrer le PID. La durée d'une boucle est de 14 milisecondes environ.

Tableau des essais:

Après toute une série de tests analysés en fonction du comportement du robot, J. Sluka a considéré ces valeurs comme un résultat satisfaisant.

TargetPower	Kp	Ki	Kd	
80	220	7	500	7111 ***
	fairly smooth, OK speed, no beeps			

Application des formules par la Méthode Ziegler-Nichols:

Type	Kp	Ki	Kd
P	0.5Kc0	0	0
PI	0.45Kc	1.2(Kp)(dT)/Pc	0
PID	0.6Kc	2(Kp)(dT)/Pc	KpPc/8/dT

Ki et Kd sont mis à zéro. Kp est égal à 100. Lancer et observer le suivi de l'ovale et les oscillations du robot.
Augmenter Kp jusqu'à ce que le système oscille d'une manière satisfaisante. Cette valeur Kp est alors considérée comme convenable pour Kc. La période Pc se détermine à partir de ce Kc.
Dans les conditions des essais, on obtient les résultats suivants par application de la méthode:
Kc = 280 ; Pc = 0,7 et dT = 0,014 sec. La Puissance "TargetPower" est limitée à 60.
Les tops sonores qu'on pourrait entendre sont utiles pour l'optimisation et le débugging.
L'idéal serait qu'aucun top ne se fasse entendre.

Type	Kp	Ki	Kd
P	0,5*280=140	0	0
PI	0,45*280=126	1,2*280*0,014/0,7= 3,...	0
PID	0,6*280=168	2*168*0,014/0,7= 6,72 soit 7	168*0,7/8/0,014=1050

En appliquant ces valeurs au programme principal, nous obtenons 3 modes de comportement du robot dans son parcours:
Les P et PI des calculs précédents donnent une bonne réponse au parcours de test ovale (test pad).
Par contre, le PID donne un parcours médiocre avec une grande quantité de tops sonores.
Après correction de "TargetPower" à 80, les K's suivants sont considérés comme acceptables:

PID	170	7	400

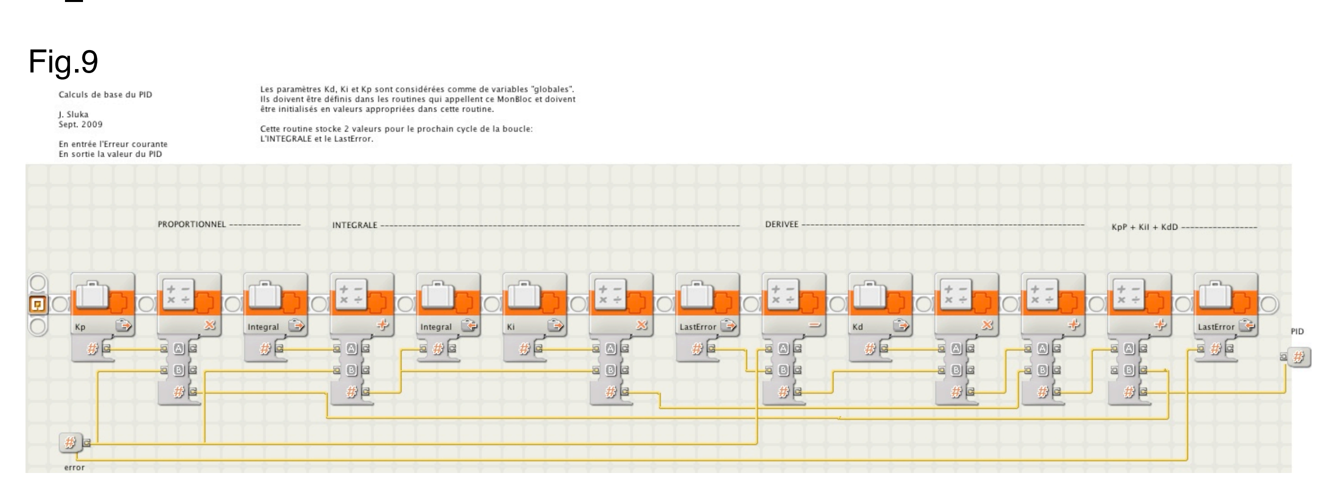
Le choix du PID complet trouve son avantage dans la puissance cible accrue, donc vitesse accrue.

Pour vous faire une idée des programmes NXT-G:
Les MonBlocs à télécharger ici:

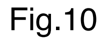
http://files.me.com/roboleo/m68hec
http://files.me.com/roboleo/68rrsl

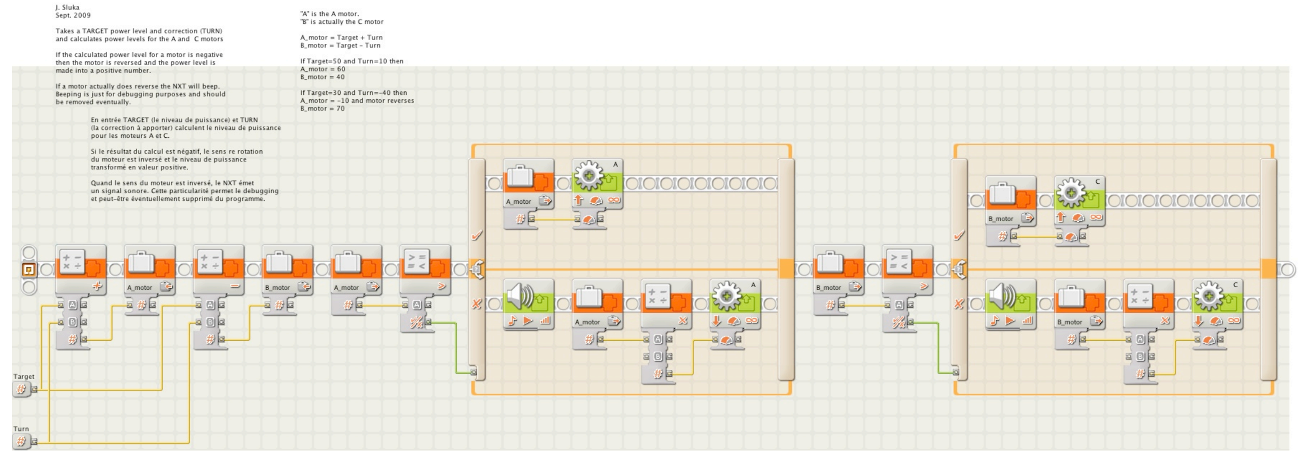
LF_PID: module de calcul

Fig.9



PID_LF_MotoControl


Fig.10

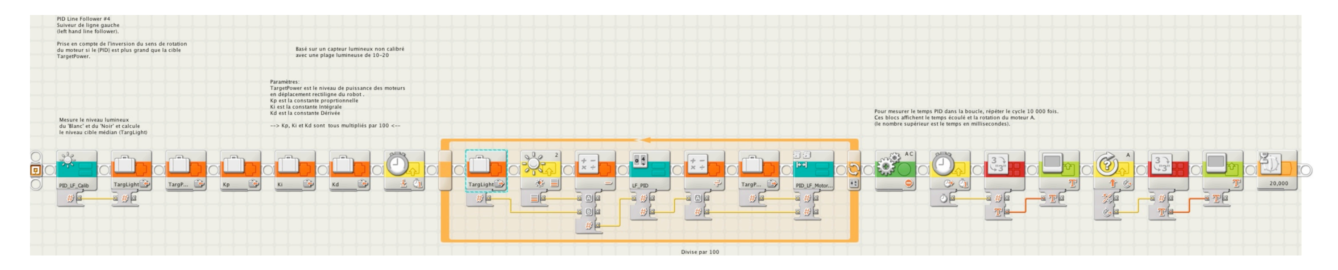


Et enfin le programme principal
A télécharger ici

http://files.me.com/roboleo/zntc6v

PID_Linefol_4

Fig.11



Nous sommes arrivés au terme de cette leçon et j'espère que vous aurez assimilé le principe du PID.

C'est un outil puissant et il faut rendre hommage à J. Sluka pour son approche en NXT-G (ce qui n'est pas du tout évident). Ses nombreux essais et tests avec commentaires m'ont permis d'en tirer quelques éléments suffisants pour illustrer cette leçon. Si le coeur vous en dit, rien ne vous empêche de tester plus finement, sachant que les résultats dépendent de chaque tracé avec ses propres conditions de luminosité. Le programme principal se prête parfaitement à toutes ces mesures et toutes les valeurs des k's.

Et maintenant, bon travail…